

intel

# Microcontroller Handbook



WILLIAM AHEARN

Order Number: 210918-003



## LITERATURE

In addition to the product line handbooks listed below, the INTEL PRODUCT GUIDE (no charge, Order No. 210846-003) provides an overview of Intel's complete product lines and customer services.

Consult the INTEL LITERATURE GUIDE (Order No. 210620) for a listing of Intel literature. TO ORDER literature in the U.S., write or call the INTEL LITERATURE DEPARTMENT, 3065 Bowers Avenue, Santa Clara, CA 95051, (800) 538-1876, or (800) 672-1833 (California only). TO ORDER literature from international locations, contact the nearest Intel sales office or distributor (see listings in the back of most any Intel literature).

Use the order blank on the facing page or call our TOLL FREE number listed above to order literature. Remember to add your local sales tax.

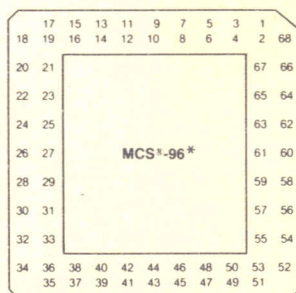
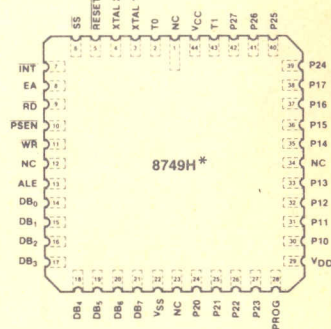
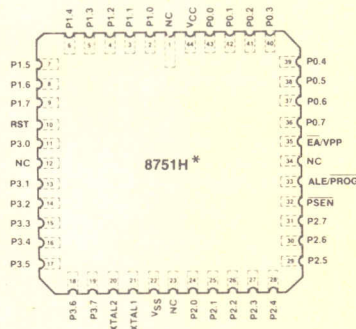
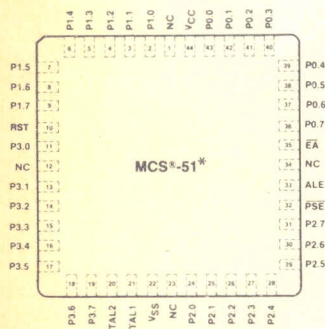
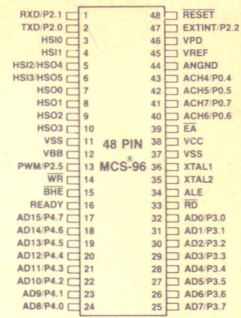
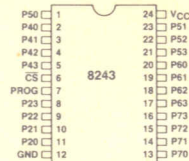
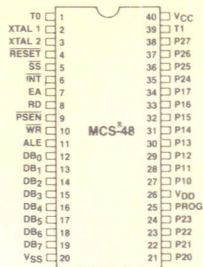
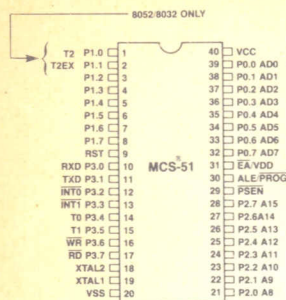
### 1985 HANDBOOKS

Product line handbooks contain data sheets, application notes, article reprints and other design information.

	* U.S. PRICE
<b>QUALITY/RELIABILITY HANDBOOK (Order No. 210997-001)</b>	<b>\$15.00</b>
Contains technical details of both quality and reliability programs and principles.	
<b>CHMOS HANDBOOK (Order No. 290005-001)</b>	<b>\$12.00</b>
Contains data sheets only on all microprocessor, peripheral, microcontroller and memory CHMOS components.	
<b>MEMORY COMPONENTS HANDBOOK (Order No. 210830-004)</b>	<b>\$18.00</b>
<b>TELECOMMUNICATION PRODUCTS HANDBOOK (Order No. 230730-003)</b>	<b>\$12.00</b>
<b>MICROCONTROLLER HANDBOOK (Order No. 210918-003)</b>	<b>\$18.00</b>
<b>MICROSYSTEM COMPONENTS HANDBOOK (Order No. 230843-002)</b>	<b>\$25.00</b>
Microprocessors and peripherals—2 Volume Set	
<b>DEVELOPMENT SYSTEMS HANDBOOK (Order No. 210940-003)</b>	<b>\$15.00</b>
<b>OEM SYSTEMS HANDBOOK (Order No. 210941-003)</b>	<b>\$18.00</b>
<b>SOFTWARE HANDBOOK (Order No. 230786-002)</b>	<b>\$12.00</b>
<b>MILITARY HANDBOOK (Order No. 210461-003)</b>	<b>\$15.00</b>
Not available until June.	
<b>COMPLETE SET OF HANDBOOKS (Order No. 231003-002)</b>	<b>\$120.00</b>
Get a 25% discount off the retail price of \$160.	

\*U.S. Price Only





MCS-96 Pin Table

PIN	SYMBOL	PIN	SYMBOL	PIN	SYMBOL	PIN	SYMBOL
1	ACH7/P0.7	18	AD0/P3.0	35	READY	52	HSI2/HS04
2	ACH6/P0.6	19	AD1/P3.1	36	T2RST/P2.4	53	HSI1
3	ACH2/P0.2	20	AD2/P3.2	37	BHE	54	HSI0
4	ACH0/P0.0	21	AD3/P3.3	38	WR	55	P1.4
5	ACH1/P0.1	22	AD4/P3.4	39	PWM/P2.5	56	P1.3
6	ACH3/P0.3	23	AD5/P3.5	40	P2.7	57	P1.2
7	NMI	24	AD6/P3.6	41	VBB	58	P1.1
8	EA	25	AD7/P3.7	42	VSS	59	P1.0
9	VCC	26	AD8/P4.0	43	HS03	60	TXD/P2.0
10	VSS	27	AD9/P4.1	44	HS02	61	RXD/P2.1
11	XTAL1	28	AD10/P4.2	45	P2.6	62	RESET
12	XTAL2	29	AD11/P4.3	46	P1.7	63	EXTINT/P2.2
13	CLKOUT	30	AD12/P4.4	47	P1.6	64	VPD
14	TEST	31	AD13/P4.5	48	P1.5	65	VREF
15	INST	32	AD14/P4.6	49	HS01	66	ANGND
16	ALE	33	AD15/P4.7	50	HS00	67	ACH4/P0.4
17	RD	34	T2CLK/P2.3	51	HSI3/HS05	68	ACH5/P0.5

\* Top View Looking Through Package





# MICROCONTROLLER HANDBOOK

1985

UP1 and a numerical suffix  
SYSTEM 2000, and UP1, and the combination of ICE, ICE, IEMX, IESC, MCS, or  
Promware, QUEST, QUEX, RipeMode, RMX, RUP1, Seamless, SOLO,  
TIBUS, MULTICHANNEL, MULTIMODULE, Plus-A-Bubble, PROMPT,  
ISDM, ISXM, Library Manager, MCS, MegaChassis, MICROMANAGER, MUL-  
tichannel, Intelligent Programming, Intelligent, IOSR, IPDS, ISEC, ISBX,  
DIS, FICE, IEMX, IEMX, Intel, Intel, Intel, Intel, Intel, Intel, Intel, Intel,  
BITBUS, COMMAND, CREDIT, Data Pipeline, GENIUS, I, ICE, ICE, IESR,

MDS is an ordering code only and is not used as a product name or trademark. MDS<sup>®</sup> is a registered trademark of Motorola Data Sciences Corporation.

\* MULTIBUS is a patented Intel bus.

Additional manuals or other Intel literature may be obtained from:

## About Our Cover:

The design on our front cover is an abstract portrayal of the basic microcontroller function. The center sphere, symbolic of a microcontroller, contains a molecular orbital diagram of the architectural construction of a cubic unit of silicon. The red pathways leading from the central sphere, are symbolic of distant or remote controlled applications.





## MICROCONTROLLER HANDBOOK

Intel Corporation makes no warranty for the use of its products and assumes no responsibility for any errors which may appear in this document nor does it make a commitment to update the information contained herein.

Intel retains the right to make changes to these specifications at any time, without notice.

Contact your local sales office to obtain the latest specifications before placing your order.

The following are trademarks of Intel Corporation and may only be used to identify Intel Products:

1985

BITBUS, COMMputer, CREDIT, Data Pipeline, GENIUS, i,  $\hat{I}$ , ICE, iCS, iDBP, iDIS, I<sup>2</sup>ICE, iLBX, i<sub>m</sub>, iMMX, Insite, Intel, intel, intelBOS, Inteleview, intelligent Identifier, intelligent Programming, Inteltec, Intellink, iOSP, iPDS, iSBC, iSBX, iSDM, iSXM, Library Manager, MCS, Megachassis, MICROMAINFRAME, MULTIBUS, MULTICHANNEL, MULTIMODULE, Plug-A-Bubble, PROMPT, Promware, QUEST, QUEX, Ripplemode, RMX/80, RUPI, Seamless, SOLO, SYSTEM 2000, and UPI, and the combination of ICE, iCS, iRMX, iSBC, MCS, or UPI and a numerical suffix.

MDS is an ordering code only and is not used as a product name or trademark. MDS\* is a registered trademark of Mohawk Data Sciences Corporation.

\* MULTIBUS is a patented Intel bus.

Additional copies of this manual or other Intel literature may be obtained from:

Intel Corporation  
Literature Department  
3065 Bowers Avenue  
Santa Clara, CA 95051



## Table of Contents

<b>ALPHANUMERIC INDEX</b> .....	iii
<b>MCS®-96 FAMILY</b>	
<b>CHAPTER 1</b>	
Introduction To MCS®-96 .....	1-1-1
<b>CHAPTER 2</b>	
Architectural Overview .....	2-1
<b>CHAPTER 3</b>	
MCS®-96 Software Design Information .....	3-1
<b>CHAPTER 4</b>	
MCS®-96 Hardware Design Information .....	4-1
<b>CHAPTER 5</b>	
MCS®-96 Data Sheet .....	5-1
<b>CHAPTER 6</b>	
MCS®-96 Article Reprint	
AR-321: High Performance Event Interface For A Microcomputer .....	6-1
<b>MCS®-51 FAMILY</b>	
<b>CHAPTER 7</b>	
MCS®-51 Architecture .....	7-1
<b>CHAPTER 8</b>	
MCS®-51 Instruction Set .....	8-1
<b>CHAPTER 9</b>	
MCS®-51 Data Sheets	
8031/8051 8031AH/8051AH 8032AH/8052AH 8751H/8751H-12 .....	9-1
8052AH-Basic .....	9-15
80C51BH/80C51BH-2 80C31BH/80C31BH-2 .....	9-24
8031AH/8051AH 8032AH/8052AH 8751H/8751H Express .....	9-39
<b>CHAPTER 10</b>	
MCS®-51 Application Notes	
AP-69: An Introduction To The Intel MCS®-51	
Single-Chip Microcomputer Family .....	10-1
AP-70: Using The Intel MCS®-51 Boolean	
Processing Capabilities .....	10-31
AP-223: 8051 Based CRT/Terminal Controller .....	10-65
<b>CHAPTER 11</b>	
MCS®-51 Article Reprint	
AR-224: Controller Chip Takes On Many	
Industrial, Computer Uses .....	11-1
<b>MCS®-48 FAMILY</b>	
<b>CHAPTER 12</b>	
MCS®-48 Single Component System .....	12-1
<b>CHAPTER 13</b>	
MCS®-48 Expanded System .....	13-1
<b>CHAPTER 14</b>	
MCS®-48 Instruction Set .....	14-1
<b>CHAPTER 15</b>	
MCS®-48 Data Sheets	
8243 .....	15-1
8048AH/8035AHL/8049AH/8039AHL/8050AH/8040AHL .....	15-7
8748H/8035H/8749H/8039H .....	15-17
MCS®-48 Express .....	15-30
80C39-9/80C49-7 .....	15-34



# Table of Contents

## THE RUP1™ FAMILY: MICROCONTROLLER WITH ON-CHIP COMMUNICATION CONTROLLER

<b>CHAPTER 1</b>	Introduction To MCS-88	16-1
<b>CHAPTER 2</b>	Architectural Overview	17-1
<b>CHAPTER 3</b>	MCS-88 Software Design Information	18-1
<b>CHAPTER 4</b>	MCS-88 Hardware Design Information	19-1
<b>CHAPTER 5</b>	MCS-88 Data Sheet	20-1
<b>CHAPTER 6</b>	MCS-88 Article Reprint	20-20
<b>CHAPTER 7</b>	MCS-88 High Performance Evaluation Kit For A Microcomputer	21-1
<b>CHAPTER 8</b>	RUP1™ Data Sheets	
	8044AH/8344AH	
	8744	
<b>CHAPTER 9</b>	RUP1™ Article Reprint	

## DESIGN CONSIDERATIONS

<b>CHAPTER 10</b>	Application Notes	22-1
	AP-125: Designing Microcontroller Systems For Electrically Noisy Environments	22-22
	AP-155: Oscillators For Microcontrollers	22-23

## DESIGN CONSIDERATIONS WHEN USING CHMOS

<b>CHAPTER 11</b>	Article Reprints	23-1
	AR-302	23-6
	AR-332	23-17

## ADVANCED PACKAGING INFORMATION

<b>CHAPTER 12</b>	AR-332: 8051 Based C8051F001 Controller	24-1
-------------------	---	------

## ALPHANUMERICAL INDEX

8031	Data Sheet	9-1
8031AH	Data Sheet	9-1
8031AH	Express Data Sheet	9-39
8032AH	Data Sheet	9-1
8032AH	Data Sheet	9-39
8035H	Data Sheet	15-7
8035AHL	Data Sheet	15-7
8039H	Data Sheet	15-7
8039AHL	Data Sheet	15-7
8040AHL	Data Sheet	15-7
8044	Application Example	19-1
8044	Architecture	17-1
8044	Serial Interface	18-1
8044AH	Data Sheet	20-1
8048AH	Data Sheet	15-7
8049AH	Data Sheet	15-7
8050AH	Data Sheet	15-7
8051	Data Sheet	9-1
8051AH	Data Sheet	9-1
8051AH	Express Data Sheet	9-39
8052AH	Data Sheet	9-1
8052AH	Basic Data Sheet	9-15
8052AH	Express Data Sheet	9-39
8243	Data Sheet	15-1
8344AH	Data Sheet	20-1
8744	Data Sheet	20-20
8748H	Data Sheet	15-17
8749H	Data Sheet	15-7
8751H	Data Sheet	9-1
8751H	Express Data Sheet	9-39
8751H-12	Data Sheet	9-1
80C31BH	Data Sheet	9-24
80C31BH-2	Data Sheet	9-24
80C39-9	Data Sheet	15-34
80C51BH	Data Sheet	9-24
80C51BH-2	Data Sheet	9-24
ADVANCED Packaging Information		24-1
Design Considerations		22-1



## ALPHANUMERICAL INDEX

## ALPHANUMERICAL INDEX

Design Considerations Application Notes . . . . .	22-2, 22-24
Design Considerations When Using CHMOS . . . . .	23-1
Design Considerations When Using CHMOS Article Reprints . . . . .	23-6, 23-17
MCS-48 Data Sheets . . . . .	15-1, 15-7, 15-17, 15-30, 15-34
MCS-48 Expanded System . . . . .	13-1
MCS-48 Instruction Set . . . . .	14-1
MCS-48 Single Component System . . . . .	12-1
MCS-51 Application Notes . . . . .	10-1, 10-31, 10-65
MCS-51 Architecture . . . . .	7-1
MCS-51 Article Reprint . . . . .	11-1
MCS-51 Data Sheets . . . . .	9-1, 9-15, 9-24, 9-39
MCS-51 Instruction Set . . . . .	8-1
MCS-96 Architectural Overview . . . . .	2-1
MCS-96 Article Reprint . . . . .	6-1
MCS-96 Data Sheet . . . . .	5-1
MCS-96 Hardware Information . . . . .	4-1
MCS-96 Introduction . . . . .	1-1
MSC-96 Software Design Information . . . . .	3-1
RUPI Data Sheets . . . . .	20-1, 20-20
RUPI Article Reprints . . . . .	21-1
RUPI-44 Family . . . . .	16-1
8024AH Basic Data Sheet . . . . .	9-15
8024AH Express Data Sheet . . . . .	9-39
8243 Data Sheet . . . . .	12-1
8344AH Data Sheet . . . . .	20-1
8744 Data Sheet . . . . .	20-20
8748H Data Sheet . . . . .	15-17
8748H Data Sheet . . . . .	15-7
8751H Data Sheet . . . . .	9-1
8751H Express Data Sheet . . . . .	9-39
8751H-15 Data Sheet . . . . .	9-1
80C31BH Data Sheet . . . . .	9-24
80C31BH-2 Data Sheet . . . . .	9-24
80C38-8 Data Sheet . . . . .	15-34
80C31BH Data Sheet . . . . .	9-24
80C31BH-5 Data Sheet . . . . .	9-24
ADVANCED Packaging Information . . . . .	24-1
Design Considerations . . . . .	22-1





# Introduction to MCS®-96

1

# CHAPTER 1

## INTRODUCTION TO MCS®-96

### 1.0 CONTINUING MICROCONTROLLER EVOLUTION

Beginning with the introduction of the world standard 8048 (MCS®-48) Microcontroller in 1976, Intel has continued to drive the evolution of single chip microcontrollers. In 1980, Intel introduced the 8051 (MCS-51) offering performance levels significantly higher than the 8048. With the advent of the 8051, the microcontroller applications base took a marked vertical leap. These versatile chips are used in applications from keyboards and terminals to controlling automobile engines. The 8051 quickly gained the position of the second generation world standard microcontroller.

Now that the semiconductor process technologies are

being pushed to new limits, it has become possible to integrate more than 100,000 transistors onto a single silicon chip. Microcontroller designers at Intel have taken today's process technology achievements and forged a new generation of single chip microcontrollers called the MCS-96. The 8096 (generic part number for MCS-96) offers the highest level of system integration ever achieved on a single chip microcontroller. It uses over 120,000 transistors to implement a high performance 16-bit CPU, 8K bytes of program memory, 232 bytes of data memory and both analog and digital types of I/O features. Figure 1-1 shows the evolution of single chip microcontroller at Intel.

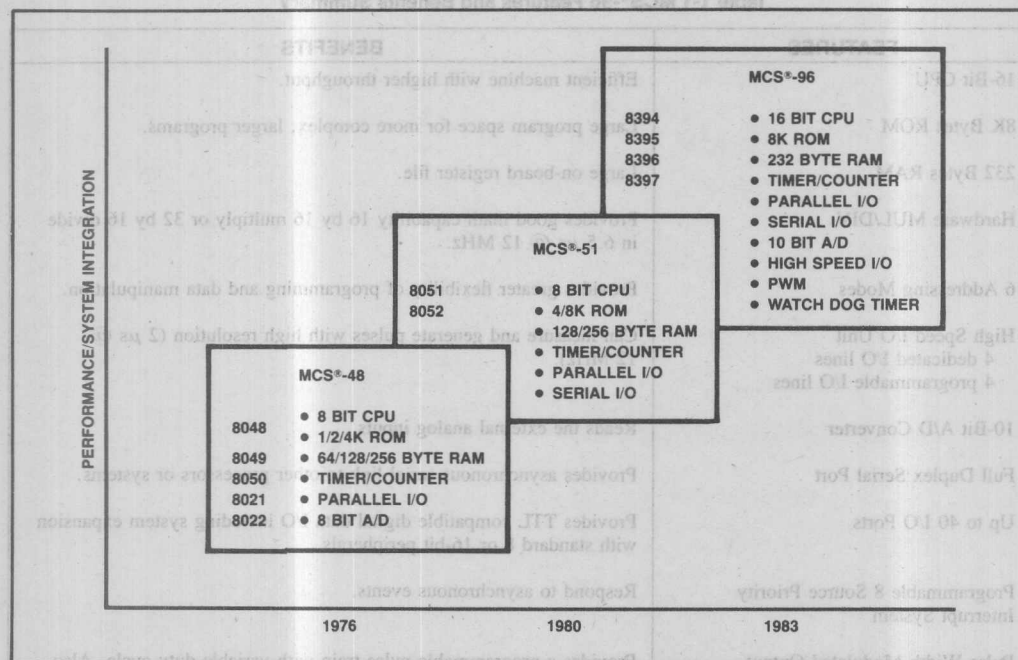


Figure 1-1. Evolution of Microcontrollers at Intel



## 1.1 INTRODUCTION TO THE MCS®-96

The 8096 consists of a 16-bit powerful CPU tightly coupled with program and data memory along with several I/O features all integrated onto a single piece of silicon. The CPU supports bit, byte, and word operations. 32-bit double words are also supported for a subset of the instruction set. With a 12 MHz input frequency, the 8096 can perform a 16-bit addition in 1.0  $\mu$ s and  $16 \times 16$  multiply or 32/16 divide in 6.5  $\mu$ s.

Four high-speed trigger inputs are provided to record the times at which external events occur with a resolution of 2  $\mu$ s (at 12 MHz crystal frequency). Up to six high-speed pulse generator outputs are provided to trigger external events at preset times. The high speed output unit can simultaneously perform timer functions, up to four such

16-bit software timers can be in operation at once in addition to the two 16-bit hardware timers.

An optional on-chip A/D converter converts up to four (in the 48-pin version) or 8 (in the 68-pin version) analog input channels into 10-bit digital values. Also provided on-chip, is a serial port, a watchdog timer, and a pulse-width modulated output signal. Table 1.1 shows the features and benefits summary for the MCS-96.

The 8096 with its 16-bit CPU and all the I/O features and interface resources on a single piece of silicon represents the highest level of system integration in the world of microcontrollers. It will open up new applications which had to use multiple chip solutions in the past.

Table 1-1 MCS®-96 Features and Benefits Summary

FEATURES	BENEFITS
16-Bit CPU	Efficient machine with higher throughput.
8K Bytes ROM	Large program space for more complex, larger programs.
232 Bytes RAM	Large on-board register file.
Hardware MUL/DIV	Provides good math capability 16 by 16 multiply or 32 by 16 divide in 6.5 $\mu$ s @ 12 MHz.
6 Addressing Modes	Provides greater flexibility of programming and data manipulation.
High Speed I/O Unit 4 dedicated I/O lines 4 programmable I/O lines	Can measure and generate pulses with high resolution (2 $\mu$ s @ 12 MHz).
10-Bit A/D Converter	Reads the external analog inputs.
Full Duplex Serial Port	Provides asynchronous serial link to other processors or systems.
Up to 40 I/O Ports	Provides TTL compatible digital data I/O including system expansion with standard 8 or 16-bit peripherals.
Programmable 8 Source Priority Interrupt System	Respond to asynchronous events.
Pulse Width Modulated Output	Provides a programmable pulse train with variable duty cycle. Also used to generate analog output.
Watchdog Timer	Provides ability to recover from software malfunction or hardware upset.
48 Pin (DIP) & 68 Pin (Flatpack, Pin Grid Array) Versions	Offers a variety of package types to choose from to better fit a specific application need for number of I/O's and package size.

## 1.2. MCS®-96 APPLICATIONS

The MCS-96 products are stand-alone high performance single chip microcontrollers designed for use in sophisticated real-time demanding applications such as industrial control, instrumentation and intelligent computer peripherals. The wide base of applications cut across all industry segments (see table 1.2). With the 16-bit CPU horsepower, high-speed math processing and high-speed I/O, the 8096 is ideal for complex motor control and axis control systems. Examples include three phase, large horsepower AC motors and robotics.

With its 10-bit A/D converter option, the device finds usage in data acquisition systems and closed-loop analog controllers. It permits considerable system integration by

combining analog and digital I/O processing in the single chip.

This chip is ideally suited in the area of instrumentation products such as gas chromatographs, which combine analog processing with high speed number crunching. The same features make it a desirable component for aerospace applications like missile guidance and control.

## 1.3. MCS®-96 FAMILY DEVELOPMENT SUPPORT TOOLS

The product family is supported by a range of Intel software and hardware development tools. These tools shorten the product development cycle, thus bringing the product to the market sooner.

### 1.3.1. MCS®-96 Software Development Package

The 8096 software development package provides development system support specifically designed for the MCS-96 family of single chip microcontrollers. The package consists of a symbolic macro assembler ASM-96, Linker/Relocator RL-96 and the librarian LIB-96. Among the high level languages, PLM-96 is offered along with a floating point math package. Additional high level languages are being developed for the MCS-96 product family.

### 1.3.2. ASM-96 MACRO Assembler

The 8096 macro assembler translates the symbolic assembly language instructions into the machine executable object code. ASM-96 enables the programmer to write the program in a modular fashion. The modular programs divide a rather complex program into smaller functional units, that are easier to code, to debug, and to change. The separate modules can then be linked and located into one program module using the RL-96 utility. This utility combines the selected input object modules into a single output object module. It also allocates memory to input segments and binds the relocatable addresses to absolute addresses. It then produces a print file that consists of a link summary, a symbol table listing and an intermediate cross-reference listing. LIB-96, another utility helps to create, modify, and examine library files. The ASM-96 runs on Intellec Series III or IV.

### 1.3.3. PL/M-96

The PL/M-96 compiler translates the PL/M-96 language into 8096 relocatable object modules. This allows improved programmer productivity and application reliability. This high level language has been efficiently designed to map into the machine architecture, so as not to trade off higher programmer productivity with inefficient code. Since the language and the compiler are optimized for the 8096 and its application environment, developing software with PL/M-96 is a 'low-risk' project.

Table 1-2 MCS®-96 Broad Base of Applications

INDUSTRIAL	
Motor Control	
Robotics	
Discrete and Continuous Process Control	
Numerical Control	
Intelligent Transducers	
INSTRUMENTATION	
Medical Instrumentation	
Liquid and Gas Chromatographs	
Oscilloscopes	
CONSUMER	
Video Recorder	
Laser Disk Drive	
High-end Video Games	
GUIDANCE & CONTROL	
Missile Control	
Torpedo Guidance Control	
Intelligent Ammunition	
Aerospace Guidance Systems	
DATA PROCESSING	
Plotters	
Color and B&W Copiers	
Winchester Disk Drive	
Tape Drives	
Impact and Non-Impact Printers	
TELECOMMUNICATIONS	
Modems	
Intelligent Line Card Control	
AUTOMOTIVE	
Ignition Control	
Transmission Control	
Anti Skid Braking	
Emission Control	



### 1.3.4. Hardware Development Support: iSBE-96

The iSBE-96 is a hardware execution and debug tool for the MCS-96 products. It consists of a monitor/debugger resident in an 8096 system. This development system interfaces with the user's 8096 system via two ribbon cables, one for the 8096 I/O ports, and the other for the memory bus. The iSBE-96 is controlled by an Intellec Series III or other computer system over a serial link. Power for the iSBE-96 can be supplied by plugging it into the MULTIBUS® card slot, or by an external power supply. The iSBE-96 is contained on one standard MULTIBUS board.

The iSBE-96 provides the most often used features for real-time hardware emulation. The user can display and modify memory, set up break points, execute with or without breakpoints and change the memory map. In addition, the user can single step through the system program.

### 1.3.5. MCS®-96 Workshop

The workshop provides the design engineer or system designer hands-on experience with the MCS-96 family of products. The course includes an explanation of the Intel 8096 architecture, system timing, input/output design. The lab sessions allow the attendees to gain in-depth knowledge of the MCS-96 product family and support tools.

### 1.3.6. Insite™ Library

The Intel Insite Library contains several application programs. A very useful program contained in the Insite is SIM-96, the software simulator for 8096. It allows software simulations of user's system. The simulator provides the ability to set breakpoints, examine and modify memory, disassemble the object code and single step through the code.

## 1.4. MCS®-96 FAMILY OF PRODUCTS

Although 8096 is the generic part number often used for the MCS-96 products throughout this manual, the product family consists of eight configurations with eight part numbers including the 8096. This wide variety of products is offered to best meet user's application requirements in terms of number of I/O's and package size. The options include on-board 8K bytes of mask programmed memory, 10-bit A/D converter, and 48 or 68 pin package type.

Table 1-3 summarizes all the current products in the MCS®-96 product family.

Table 1-3 MCS®-96 Family of Products

OPTIONS		68 PIN	48 PIN
DIGITAL I/O	ROMLESS	8096	8094
	ROM	8396	8394
ANALOG AND DIGITAL I/O	ROMLESS	8097	8095
	ROM	8397	8395

The 48 pin version is available in a DIP (dual inline) package.

The 68 pin version comes in two packages, the Plastic Flatpack and the Pin Grid Array.

---

# Architectural Overview

2

---





## CHAPTER 2 ARCHITECTURAL OVERVIEW

### 2.0. INTRODUCTION

The 8096 can be separated into several sections for the purpose of describing its operation. There is a CPU, a programmable High Speed I/O Unit, an analog to digital converter, a serial port, and a Pulse Width Modulated (PWM) output for digital to analog conversion. In addition to these functional units, there are some sections which support overall operation of the chip such as the clock generator and the back-bias generator. The CPU and the programmable I/O make the 8096 very different from any other microcontroller, let us first examine the CPU.

### 2.1. CPU OPERATION

The major components of the CPU on the 8096 are the Register File and the RALU. Communication with the outside world is done through either the Special Function Registers (SFRs) or the Memory Controller. The RALU (Register/Arithmetic Logic Unit) does not use an accumulator, it operates directly on the 256-byte register space made up of the Register File and the SFRs. Efficient I/O

operations are possible by directly controlling the I/O through the SFRs. The main benefits of this structure are the ability to quickly change context, the absence of accumulator bottleneck, and fast throughput and I/O times.

#### 2.1.1. CPU Buses

A "Control Unit" and two buses connect the Register File and RALU. Figure 2-1 shows the CPU with its major bus connections. The two buses are the "A-Bus" which is 8-bits wide, and the "D-Bus" which is 16-bits wide. The D-Bus transfers data only between the RALU and the Register File or Special Function Registers (SFRs). The A-Bus is used as the address bus for the above transfers or as a multiplexed address/data bus connecting to the "Memory Controller". Any accesses of either the internal ROM or external memory are done through the Memory Controller.

Within the memory controller is a slave program counter (Slave PC) which keeps track of the PC in the CPU. By having most program fetches from memory referenced to

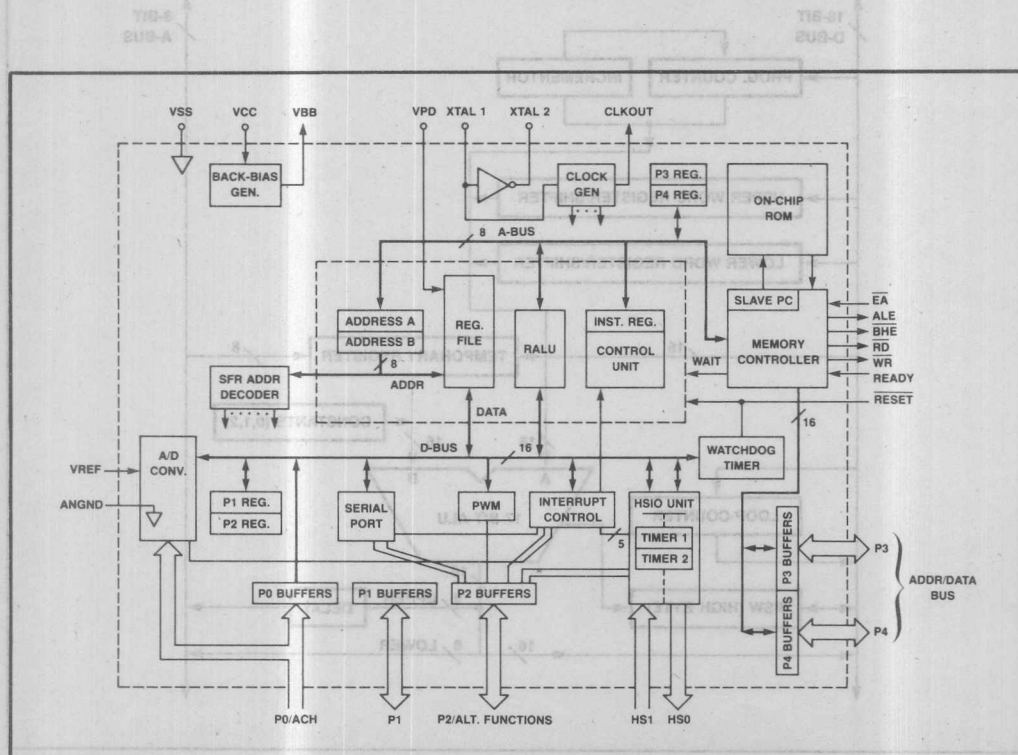


Figure 2-1. Block Diagram (For simplicity, lines connecting port registers to port buffers are not shown.)

the slave PC, the processor saves time as addresses seldom have to be sent to the memory controller. If the address jumps sequence then the slave PC is loaded with a new value and processing continues. Data fetches from memory are also done through the memory controller, but the slave PC is bypassed for this operation.

## 2.1.2. CPU Register File

The Register File contains 232 bytes of RAM which can be accessed as bytes, words, or double-words. Since each of these locations can be used by the RALU, there are essentially 232 "accumulators". The first word in the Register File is reserved for use as the stack pointer so it can not be used for data when stack manipulations are taking place. Addresses for accessing the Register File and SFRs are temporarily stored in two 8-bit address registers by the CPU hardware.

## 2.1.3. RALU Control

Instructions to the RALU are taken from the A-Bus and stored temporarily in the instruction register. The Control

Unit decodes the instructions and generates the correct sequence of signals to have the RALU perform the desired function. Figure 2-1 shows the instruction register and the control unit.

## 2.1.4. RALU

Most calculations performed by the 8096 take place in the RALU. The RALU, shown in Figure 2-2, contains a 17-bit ALU, the Program Status Word (PSW), the Program Counter (PC), a loop counter, and three temporary registers. All of the registers are 16-bits or 17-bits (16 + sign extension) wide. Some of the registers have the ability to perform simple operations to off-load the ALU.

A separate incrementer is used for the PC; however, jumps must be handled through the ALU. Two of the temporary registers have their own shift logic. These registers are used for the operations which require logical shifts, including Normalize, Multiply, and Divide. The "Lower Word" register is used only when double-word quantities are being shifted, the "Upper Word" register is used

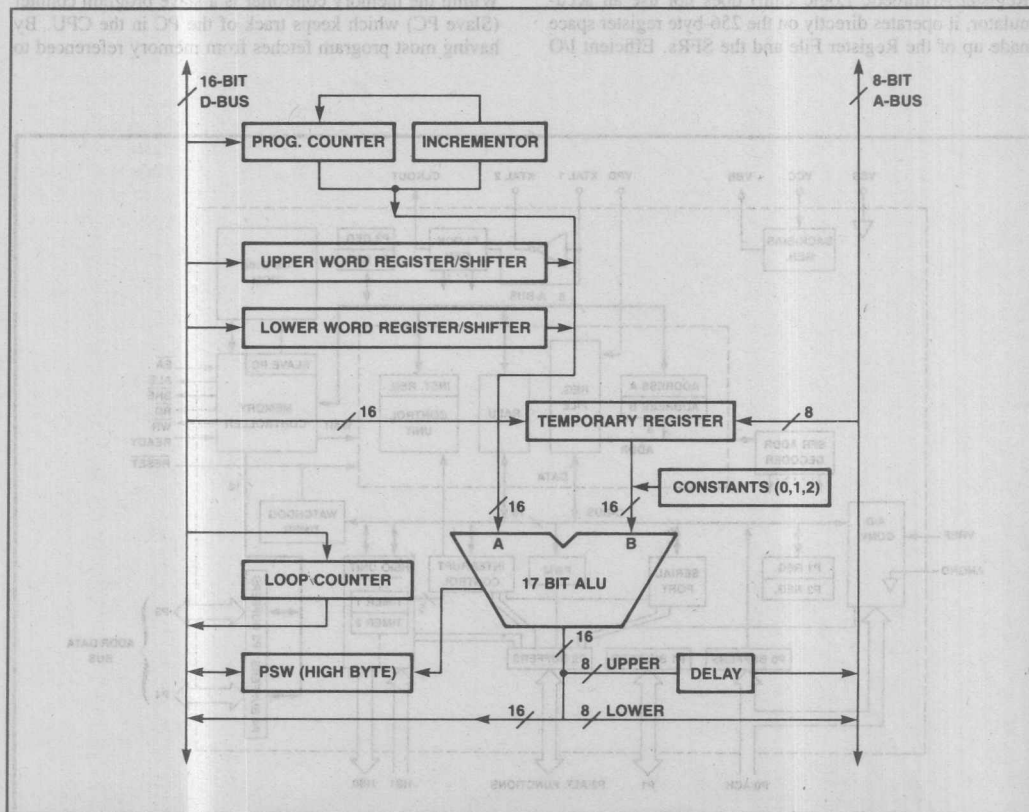


Figure 2-2. RALU Block Diagram

whenever a shift is performed or as a temporary register for many instructions. Repetitive shifts are counted by the 5-bit "Loop Counter".

A temporary register is used to store the second operand of two operand instructions. This includes the multiplier during multiplications and the divisor during divisions. To perform subtractions, the output of this register can be complemented before being placed into the "B" input of the ALU.

The DELAY shown in Figure 2-2 is used to convert the 16-bit bus into an 8-bit bus. This is required as all addresses and instructions are carried on the 8-bit A bus. Several constants, such as 0, 1 and 2 are stored in the RALU for use in speeding up certain calculations. These come in handy when the RALU needs to make a 2's complement number or perform an increment or decrement instruction.

## 2.2. BASIC TIMING

The 8096 requires an input clock frequency of between 6.0 MHz and 12 MHz to function. This frequency can be applied directly to XTAL1. Alternatively, since XTAL1 and XTAL2 are inputs and outputs of an inverter, it is also possible to use a crystal to generate the clock. A block diagram of the oscillator section is shown in Figure 2-3. Details of the circuit and suggestions for its use can be found in section 4.1.

### 2.2.1. Internal Timings

The crystal or external oscillator frequency is divided by

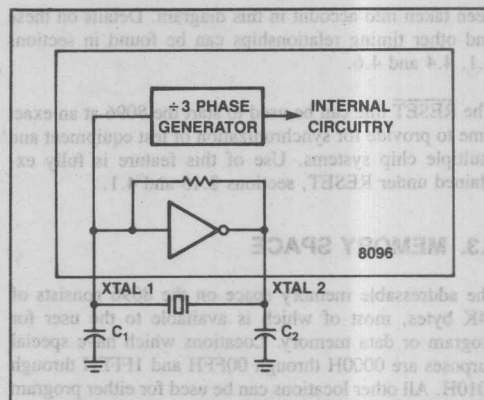


Figure 2-3. Block Diagram of Oscillator

3 to generate the three internal timing phases as shown in Figure 2-4. Each of the internal phases repeat every 3 oscillator periods: 3 oscillator periods are referred to as one "state time", the basic time measurement for 8096 operations. Most internal operations are synchronized to either Phase A, B or C, each of which have a 33% duty cycle. Phase A is represented externally by CLKOUT, a signal available on the 68-pin part. Phases B and C are not available externally. The relationships of XTAL1, CLKOUT, and Phases A, B, and C are shown in Figure 2-4. It should be noted that propagation delays have not

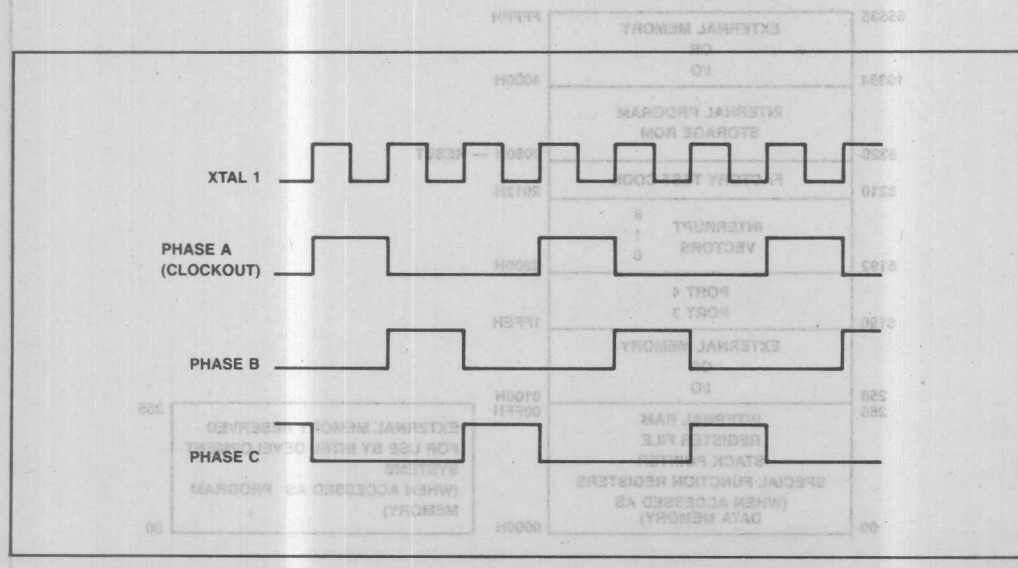


Figure 2-4. Internal Timings Relative to XTAL 1



been taken into account in this diagram. Details on these and other timing relationships can be found in sections 4.1, 4.4 and 4.6.

The **RESET** line can be used to start the 8096 at an exact time to provide for synchronization of test equipment and multiple chip systems. Use of this feature is fully explained under **RESET**, sections 2.15 and 4.1.

### 2.3. MEMORY SPACE

The addressable memory space on the 8096 consists of 64K bytes, most of which is available to the user for program or data memory. Locations which have special purposes are 0000H through 00FFH and 1FFEH through 2010H. All other locations can be used for either program or data storage or for memory mapped peripherals. A memory map is shown in figure 2-5.

### 2.3.1. Register File

Locations 00H through 0FFH contain the Register File and SFRs. Complete information on this section of memory space can be found in section 2.4. No code can be executed from this internal RAM section. If an attempt to execute instructions from locations 000H through 0FFH is made, the instructions will be fetched from *external* memory. This section of external memory is reserved for use by Intel development tools. Execution of a nonmaskable interrupt (NMI) will force a call to external location

0000H, therefore, the NMI instruction is also reserved for Intel development tools.

### 2.3.2. Reserved Memory Spaces

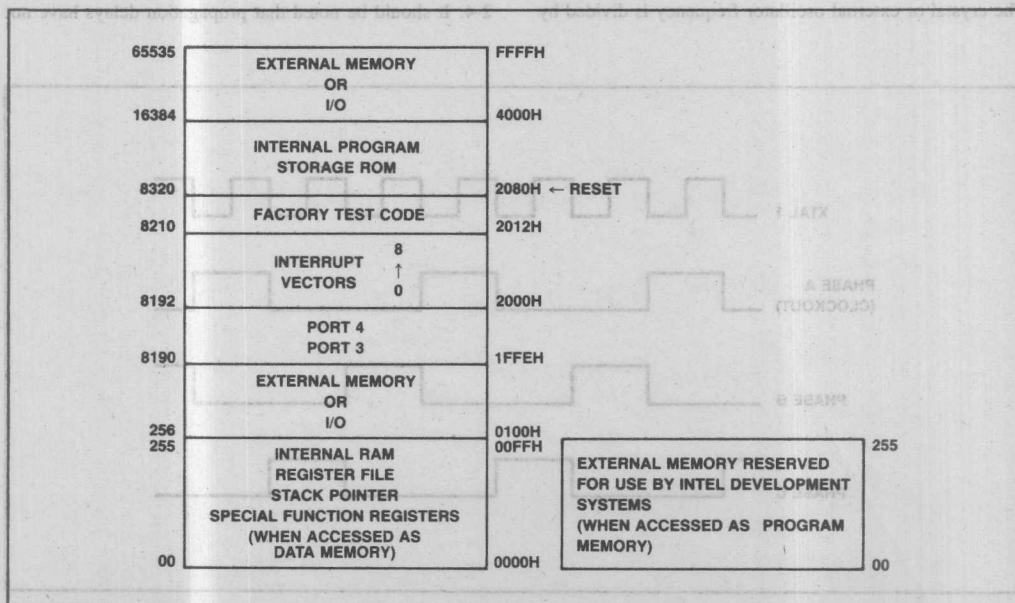
Locations 1FFEh and 1FFFh are reserved for Ports 3 and 4 respectively. This is to allow easy reconstruction of these ports if external memory is used in the system. An example of reconstructing the I/O ports is given in section 4.6.7. If ports 3 and 4 are not going to be reconstructed then these locations can be treated as any other external memory location.

The 9 interrupt vectors are stored in locations 2000H through 2011H. The 9th vector is used by Intel development systems, as explained in section 2.5. Internal locations 2012H through 207FH are reserved for Intel's factory test code. To ensure compatibility with future parts external locations 2012H through 207FH must contain the hex value FFH.

Resetting the 8096 causes instructions to be fetched starting from location 2080H. This location was chosen to allow a system to have up to 8K of RAM continuous with the register file. Further information on reset can be found in section 2.15.

### 2.3.3. Internal ROM

When a ROM part is ordered, the internal memory locations 2080H through 3FFFH are user specified as are the interrupt vectors in locations 2000H through 2011H.



**Figure 2-5. Memory Map**

## ARCHITECTURAL OVERVIEW

Instruction and data fetches from the internal ROM occur only if the part has a ROM, EA is tied high, and the address is between 2000H and 3FFFH. At all other times data is accessed from either the internal RAM space or external memory and instructions are fetched from external memory.

### 2.3.4. Memory Controller

The RALU talks to the memory (except for the locations in the register file and SFR space) through the memory controller which is connected to the RALU by the A-bus and several control lines. Since the A-bus is eight bits wide, the memory controller uses a Slave Program Counter to avoid having to always get the instruction location from the RALU. This slave PC is incremented after each fetch. When a jump or call occurs, the slave PC must be loaded from the A-bus before instruction fetches can continue.

In addition to holding a slave PC, the memory controller contains a 3 byte queue to help speed execution. This queue is transparent to the RALU and to the user unless wait states are forced during external bus cycles. The instruction execution times shown in Tables 3-3 and 3-4 show the normal execution times with no wait states

added. Reloading the slave PC and fetching the first byte of the new instruction stream takes 4 state times. This is reflected in the jump taken/not-taken times shown in Table 3-4.

### 2.3.5. System Bus

External memory is addressed through lines AD0 through AD15 which form a 16-bit multiplexed (address/data) data bus. These lines share pins with I/O ports 3 and 4. The falling edge of the Address Latch Enable (ALE) line is used to provide a clock to a transparent latch (74LS373) in order to demultiplex the bus. A typical circuit and the required timings are shown in section 4.6. Since the 8096's external memory can be addressed as either bytes or words, the decoding is controlled with two lines, Bus High Enable (BHE) and Address/Data Line 0 (AD0). The BHE line must be transparently latched, just as the addresses are.

To avoid confusion during the explanation of the memory system it is reasonable to give names to the demultiplexed address/data signals. The address signals will be called MA0 through MA15 (Memory Address), and the data signals will be called MD0 through MD15 (Memory Data).

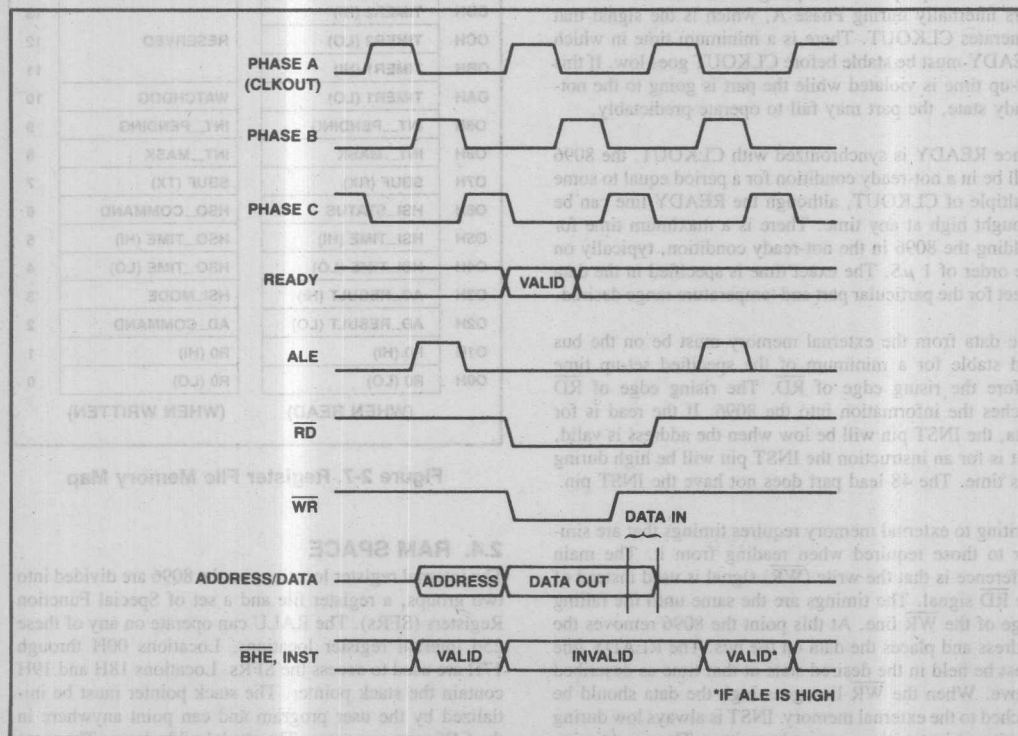


Figure 2-6. External Memory Timings

When  $\overline{\text{BHE}}$  is active (low), the memory connected to the high byte of the data bus should be selected. When  $\text{MA0}$  is low the memory connected to the low byte of the data bus should be selected. In this way accesses to a 16-bit wide memory can be to the low (even) byte only ( $\text{MA0}=0$ ,  $\overline{\text{BHE}}=1$ ), to the high (odd) byte only ( $\text{MA0}=1$ ,  $\overline{\text{BHE}}=0$ ), or to both bytes ( $\text{MA0}=0$ ,  $\overline{\text{BHE}}=0$ ). When a memory block is being used only for reads,  $\overline{\text{BHE}}$  and  $\text{MA0}$  need not be decoded.

Figure 2-6 shows the idealized waveforms related to the following description of external memory manipulations. For exact timing specifications please refer to the latest data sheet. When an external memory fetch begins, the address latch enable (ALE) line rises, the address is put on  $\text{AD0-AD15}$  and  $\overline{\text{BHE}}$  is set to the required state. ALE then falls, the address is taken off the pins, and the RD (Read) signal goes low. The READY line can be pulled low to hold the processor in this condition for a few extra state times.

## 2.3.6. Bus Control Lines

The READY line can be used to hold the processor in the above condition in order to allow access to slow memories or for DMA purposes. Sampling of the READY line occurs internally during Phase A, which is the signal that generates CLKOUT. There is a minimum time in which READY must be stable before CLKOUT goes low. If this set-up time is violated while the part is going to the not-ready state, the part may fail to operate predictably.

Since READY is synchronized with CLKOUT, the 8096 will be in a not-ready condition for a period equal to some multiple of CLKOUT, although the READY line can be brought high at any time. There is a maximum time for holding the 8096 in the not-ready condition, typically on the order of 1  $\mu\text{S}$ . The exact time is specified in the data sheet for the particular part and temperature range desired.

The data from the external memory must be on the bus and stable for a minimum of the specified set-up time before the rising edge of RD. The rising edge of RD latches the information into the 8096. If the read is for data, the INST pin will be low when the address is valid, if it is for an instruction the INST pin will be high during this time. The 48-lead part does not have the INST pin.

Writing to external memory requires timings that are similar to those required when reading from it. The main difference is that the write ( $\overline{\text{WR}}$ ) signal is used instead of the RD signal. The timings are the same until the falling edge of the  $\overline{\text{WR}}$  line. At this point the 8096 removes the address and places the data on the bus. The READY line must be held in the desired state at that time as described above. When the  $\overline{\text{WR}}$  line goes high the data should be latched to the external memory. INST is always low during a write, as instructions cannot be written. The exact timing specifications for memory accesses can be found in the data sheet.

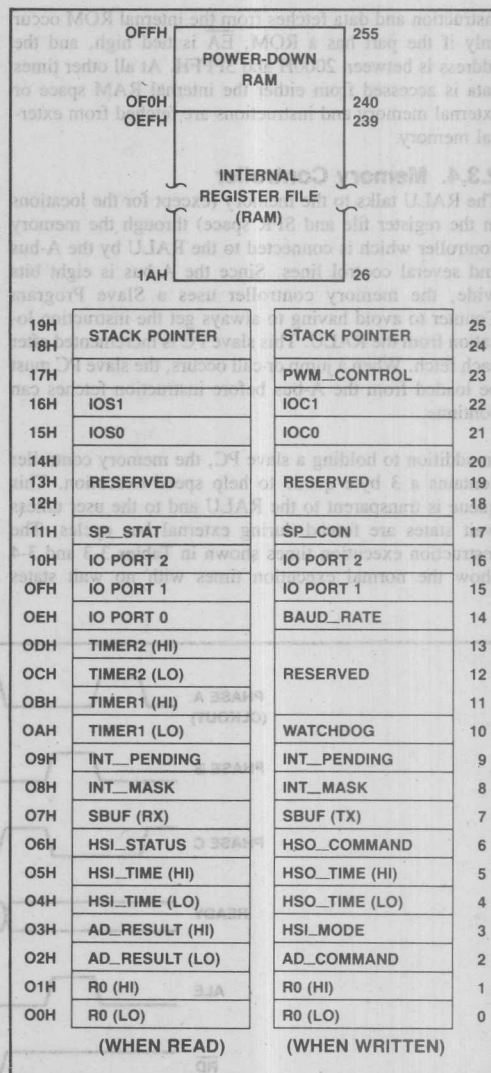


Figure 2-7. Register File Memory Map

## 2.4. RAM SPACE

The internal register locations on the 8096 are divided into two groups, a register file and a set of Special Function Registers (SFRs). The RALU can operate on any of these 256 internal register locations. Locations 00H through 17H are used to access the SFRs. Locations 18H and 19H contain the stack pointer. The stack pointer must be initialized by the user program and can point anywhere in the 64K memory space. The stack builds down. There are no restrictions on the use of the remaining 230 locations except that code cannot be executed from them.



## 2.4.1. Special Function Registers

All of the I/O on the 8096 is controlled through the SFRs. Many of these registers serve two functions; one if they

shows the locations and names of these registers. A summary of the capabilities of each of these registers is shown

Register	Description	Chapter
R0	Zero Register — Always reads as a zero, useful for a base when indexing and as a constant for calculations and compares.	3.2.7
AD _ RESULT	A/D Result Hi/Low — Low and high order Results of the A/D converter (byte read only)	2.9.3
AD _ COMMAND	A/D Command Register — Controls the A/D	2.9.2
HSI _ MODE	HSI Mode Register — Sets the mode of the High Speed Input unit.	2.7.1
HSI _ TIME	HSI Time Hi/Lo — Contains the time at which the High Speed Input unit was triggered. (word read only)	2.7.4
HSO _ TIME	HSO Time Hi/Lo — Sets the time for the High Speed Output to execute the command in the Command Register. (word write only)	2.8.3
HSO _ COMMAND	HSO Command Register — Determines what will happen at the time loaded into the HSO Time registers.	2.8.2
HSI _ STATUS	HSI Status Registers — Indicates which HSI pins were detected at the time in the HSI Time registers.	2.7.4
SBUF (TX)	Transmit buffer for the serial port, holds contents to be outputed.	2.11
SBUF (RX)	Receive buffer for the serial port, holds the byte just received by the serial port.	2.11
INT _ MASK	Interrupt Mask Register — Enables or disables the individual interrupts.	2.5.2 3.6.2
INT _ PENDING	Interrupt Pending Register — Indicates when an interrupt signal has occurred on one of the sources.	2.5.2 3.6.2
WATCHDOG	Watchdog Timer Register — Written to periodically to hold off automatic reset every 64K state times.	2.14
TIMER1	Timer 1 Hi/Lo — Timer 1 high and low bytes. (word read only)	2.6.1 2.7-8
TIMER2	Timer 2 Hi/Lo — Timer 2 high and low bytes. (word read only)	2.6.2 2.7-8
IOPORT0	Port 0 Register — Levels on pins of port 0.	2.12.1
BAUD _ RATE	Register which contains the baud rate, this register is loaded sequentially.	2.11.4
IOPORT1	Port 1 Register — Used to read or write to Port 1.	2.12.2
IOPORT2	Port 2 Register — Used to read or write to Port 2.	2.12.3
SP _ STAT	Serial Port Status — Indicates the status of the serial port.	2.11.3
SP _ CON	Serial port control — Used to set the mode of the serial port.	2.11.1
IOS0	I/O Status Register 0 — Contains information on the HSO status.	2.13.4
IOS1	I/O Status Register 1 — Contains information on the status of the timers and of the HSI.	2.13.5 3.7.2
IOC0	I/O Control Register 0 — Controls alternate functions of HSI pins, Timer 2 reset sources and Timer 2 clock sources.	2.13.2
IOC1	I/O Control Register 1 — Controls alternate functions of Port 2 pins, timer interrupts and HSI interrupts.	2.13.3
PWM _ CONTROL	Pulse Width Modulation Control Register — Sets the duration of the PWM pulse.	2.10 4.3.2
OPORT2	Port 2 Register — Used to read or write to Port 2.	2.12.3

Figure 2-8. SFR Summary

in Figure 2-8, with complete descriptions reserved for later chapters. Note that these registers can be accessed only as bytes unless otherwise indicated.

Within the SFR space are several registers labeled as "RESERVED". These registers are reserved for future expansion or test purposes. Reads or writes of these registers may produce unexpected results. For example, writing to location 000CH will set both timers to 0FFFXH, this feature is for use in testing the part and should not be used in programs.

## 2.4.2. Power Down

The upper 16 RAM locations (0F0H through 0FFH) receive their power from both the VCC pin and the VPD pin. If it is desired to keep the memory in these locations alive during a power down situation, one need only keep voltage on the VPD pin. The current required to keep the RAM alive is approximately 1 milliamp (refer to the data sheet for the exact specification).

To place the 8096 into a power down mode, the RESET pin is pulled low. Two state times later the part will be in reset. This is necessary to prevent the part from writing into RAM as the power goes down. The power may now be removed from the VCC pin, the VPD pin must remain within specifications. The 8096 can remain in this state for any amount of time and the 16 RAM bytes will retain their values.

To bring the 8096 out of power down, RESET is held low while VCC is applied. Two state times after the oscillator and the back bias generator have stabilized (~1 millisecond), the RESET pin can be pulled high. The 8096 will begin to execute code at location 02080H 10 state times after RESET is pulled high. Figure 2-9 shows a timing diagram of the power down sequence. To ensure that the 2 state time minimum reset time (synchronous with CLKOUT) is met, it is recommended that 10 XTAL1

cycles be used. Suggestions for actual hardware connections are given in section 4.1. Reset is discussed in section 2.15.

## 2.5. INTERRUPT STRUCTURE

### 2.5.1. Interrupt Sources

Eight interrupt sources are available on the 8096. When enabled, an interrupt occurring on any of these sources will force a call to the location stored in the vector location for that source. The interrupt sources and their respective vector locations are listed in Figure 2-10. In addition to the 8 standard interrupts, there is a TRAP instruction which acts as a software generated interrupt. This instruction is not currently supported by the MCS-96 Assembler and is reserved for use by Intel development systems. Many of the interrupt sources can be activated by several methods, Figure 2-11 shows all of the possible sources for interrupts.

Source	Vector Location		Priority
	(High Byte)	(Low Byte)	
Software	2011H	2010H	Not Applicable
Extint	200FH	200EH	
Serial Port	200DH	200CH	7 (Highest)
Software Timers	200BH	200AH	6
HSI.0	2009H	2008H	5
High Speed	2007H	2006H	4
Outputs			3
HSI Data Available	2005H	2004H	2
A/D Conversion Complete	2003H	2002H	1
Timer Overflow	2001H	2000H	0 (Lowest)

Figure 2-10. Interrupt Vector Locations

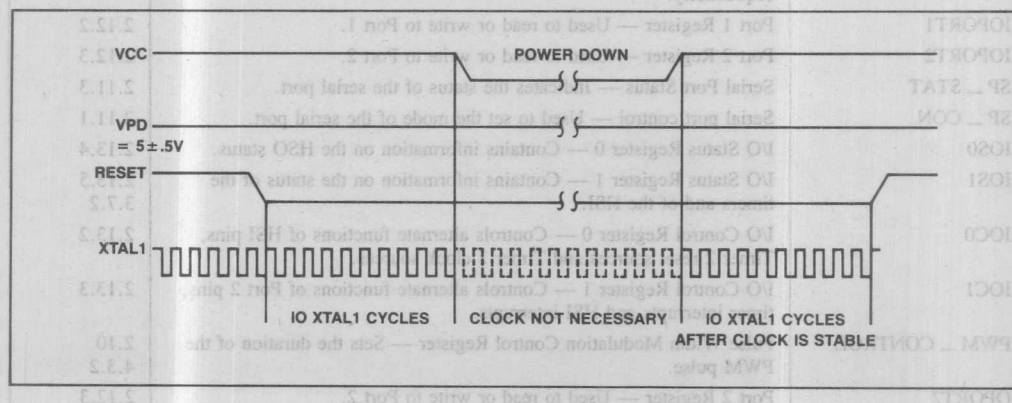


Figure 2-9. Power Down Timing

## 2.5.2. Interrupt Control

A block diagram of the interrupt system is shown in Figure 2-12. Each of the interrupt sources is tested for a 0 to 1 transition. If this transition occurs, the corresponding bit in the Interrupt Pending Register, located at 0009H, is set. The bit is cleared when the vector is taken to the interrupt routine. Since this register can be written to, it is possible to generate software interrupts by setting bits within the register, or remove pending interrupts by clearing the bits in this register. The pending register can be set even if the interrupt is disabled.

Caution must be used when writing to the pending register to clear interrupts. If the interrupt has already been acknowledged when the bit is cleared, a 4 state time "partial" interrupt cycle will occur. This is because the 8096 will have to fetch the next instruction of the normal instruction flow, instead of proceeding with the interrupt processing as it was going to. The effect on the program will be essentially that of an extra NOP. This can be prevented by clearing the bits using a 2 operand immediate logical, as the 8096 holds off acknowledging interrupts during these "read/modify/write" instructions.

Enabling and disabling of individual interrupts is done through the Interrupt Mask Register, located at 0008H. If the bit in the mask register is a 1 then the interrupt is enabled, otherwise it is disabled. Even if an interrupt is masked it may still become pending. It may, therefore, be desirable to clear the pending bit before unmasking an interrupt.

The Interrupt Mask Register is also the low byte of the PSW. All of the interrupts may be enabled and disabled simultaneously by using the "EI" (Enable Interrupt) and "DI" (Disable Interrupt) instructions. EI and DI set and clear PSW.9, the interrupt enable bit, they do not effect the contents of the mask register.

## 2.5.3. Interrupt Priority Programming

The priority encoder looks at all of the interrupts which are *both pending and enabled*, and selects the one with the highest priority. The priorities are shown in Figure 2-10 (7 is highest, 0 is lowest.) The interrupt generator then forces a call to the location in the indicated vector location. This location would be the starting location of the Interrupt Service Routine (ISR).

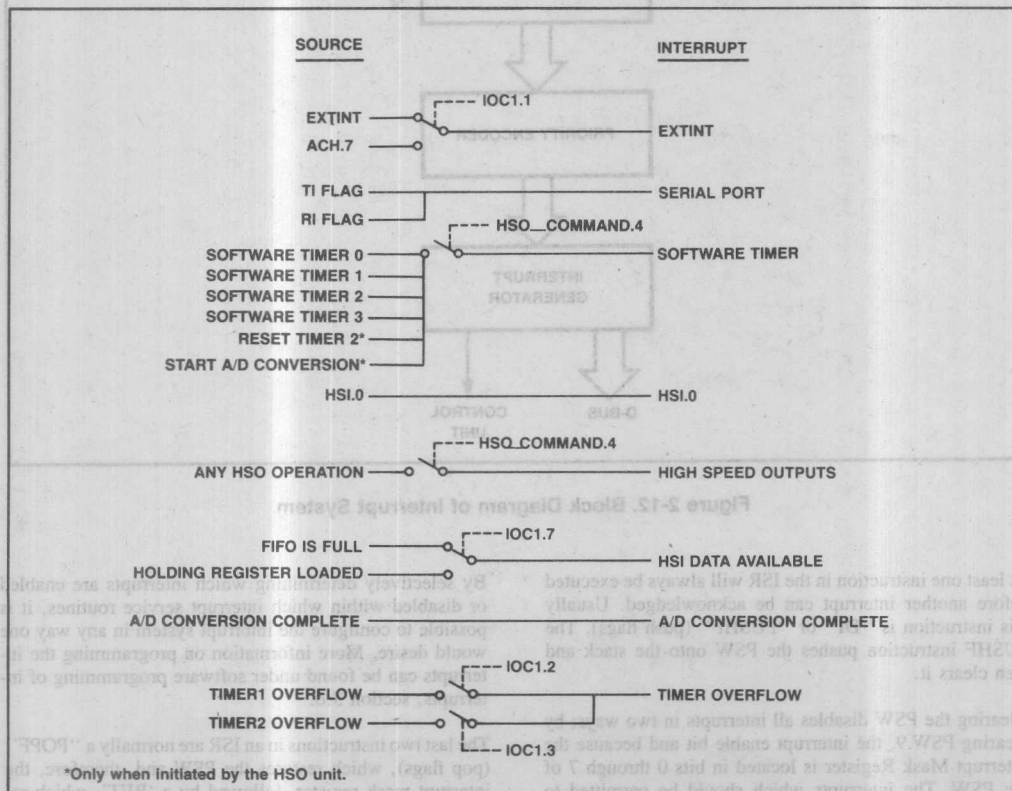


Figure 2-11. All Possible Interrupt Sources



## ARCHITECTURAL OVERVIEW

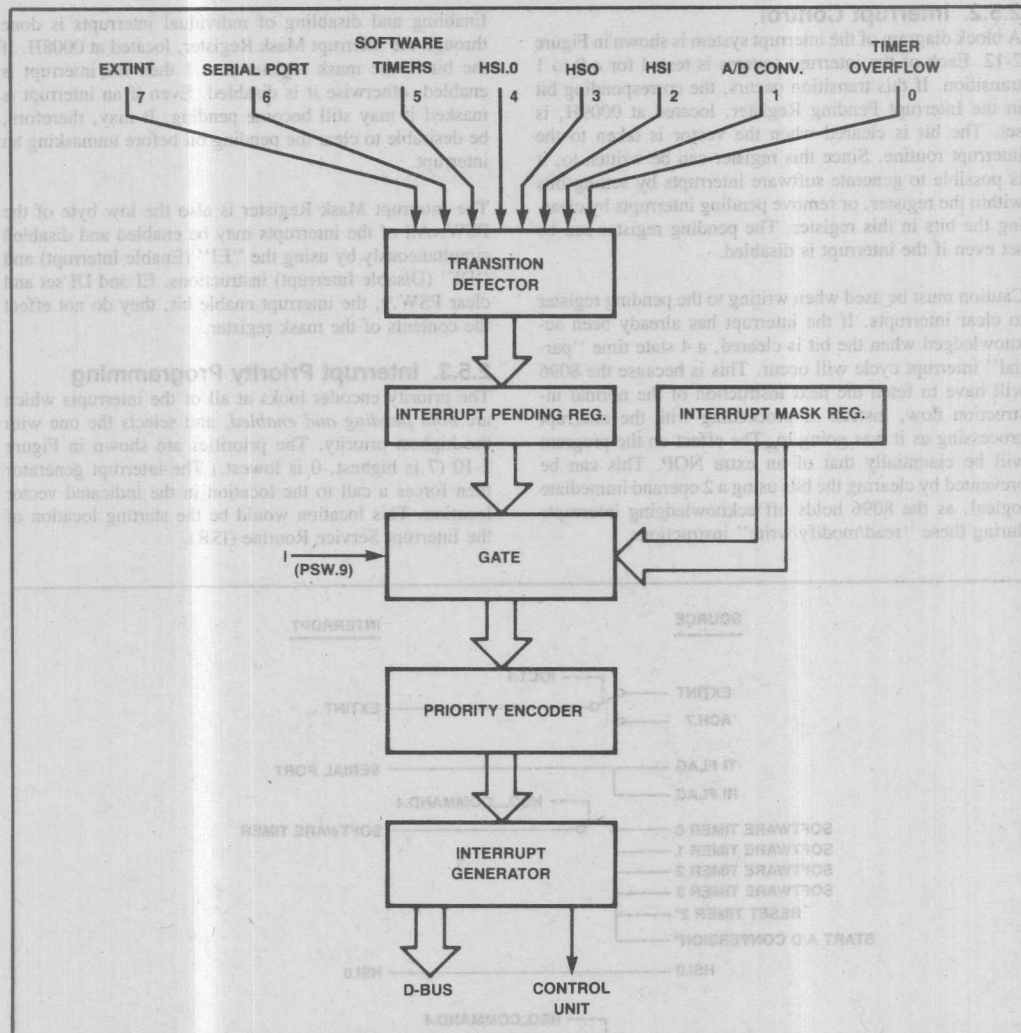


Figure 2-12. Block Diagram of Interrupt System

At least one instruction in the ISR will always be executed before another interrupt can be acknowledged. Usually this instruction is "DI" or "PUSHF" (push flags). The PUSHF instruction pushes the PSW onto the stack and then clears it.

Clearing the PSW disables all interrupts in two ways; by clearing PSW.9, the interrupt enable bit and because the Interrupt Mask Register is located in bits 0 through 7 of the PSW. The interrupts which should be permitted to interrupt this ISR can then be set in the mask register and an "EI" instruction executed.

By selectively determining which interrupts are enabled or disabled within which interrupt service routines, it is possible to configure the interrupt system in any way one would desire. More information on programming the interrupts can be found under software programming of interrupts, section 3.6.

The last two instructions in an ISR are normally a "POPF" (pop flags), which restores the PSW and, therefore, the interrupt mask register, followed by a "RET", which restores the Program Counter. Execution will then continue from the point at which the call was forced.

## 2.5.4. Interrupt Timing

Interrupts are not always acknowledged immediately. If the interrupt signal does not occur prior to 4 state-times before the end of an instruction, the interrupt will not be acknowledged until after the next instruction has been executed. This is because an instruction is fetched and prepared for execution a few state times before it is actually executed.

There are 6 instructions which always inhibit interrupts from being acknowledged until after the next instruction has been executed. These instructions are:

EI, DI	— Enable and Disable Interrupts
POPF, PUSHF	— Pop and Push Flags
SIGND	— Prefix to perform signed multiply and divide (Note that this is not an ASM-96 Mnemonic, but is used for signed multiply and divide)
TRAP	— Software interrupt

When an interrupt is acknowledged, a call is forced to the location indicated by the specified interrupt vector. This call occurs after the completion of the instruction in process, except as noted above. The procedure of getting the vector and forcing the call requires 21 state times. If the stack is in external RAM an additional 3 state times are required.

The maximum number of state times required from the time an interrupt is generated (not acknowledged) until the 8096 begins executing code at the desired location is the time of the longest instruction, NORML (Normalize — 43 state times), plus the 4 state times prior to the end of the previous instruction, plus the response time (21 to 24 state times). Therefore, the maximum response time is 71 (43 + 4 + 24) state times. This does not include the 12 state times required for PUSHF if it is used as the first instruction in the interrupt routine or additional latency caused by having the interrupt masked or disabled.

Interrupt latency time can be reduced by careful selection of instructions in areas of code where interrupts are expected. Using 'EI' followed immediately by a long instruction (e.g. MUL, NORML, etc.) will increase the maximum latency by 4 state times, as an interrupt cannot occur between EI and the instruction following EI. The 'DI', 'PUSHF', 'POPF' and 'TRAP' instructions will also cause the same situation. Typically the PUSHF, POPF and TRAP instructions would only effect latency when one interrupt routine is already in process, as these instructions are seldom used at other times.

## 2.6. TIMERS

Two 16-bit timers are available for use on the 8096. The first is designated 'Timer 1', the second, 'Timer 2'. Timer 1 is used to synchronize events to real time, while Timer 2 can be clocked externally and synchronizes events to external occurrences.

### 2.6.1. Timer 1

Timer 1 is clocked once every eight state times and can be cleared only by executing a reset. The only other way to change its value is by writing to 000CH but this is a test mode which sets both timers to 0FFFFH and should not be used in programs.

### 2.6.2. Timer 2

Timer 2 can be incremented by transitions (one count each transition, rising and falling) on either T2CLK or HSI.1. The multiple functionality of the timer is determined by the state of I/O Control Register 0, bit 7 (IOC0.7). To ensure that all CAM entries are checked each count of Timer 2, the maximum transition speed is limited to once per eight state times. Timer 2 can be cleared by: executing a reset, by setting IOC0.1, by triggering HSO channel 0EH, or by pulling T2RST or HSI.0 high. The HSO and

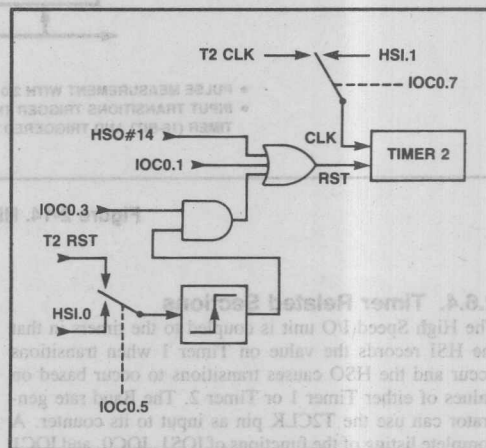


Figure 2-13. Timer 2 Clock and Reset Options

CAM are described in section 2.8. IOC0.3 and IOC0.5 control the resetting of Timer 2. Figure 2-13 shows the different ways of manipulating Timer 2.

### 2.6.3. Timer Interrupts

Both Timer 1 and Timer 2 can be used to trigger a timer overflow interrupt and set a flag in the I/O Status Register 1 (IOS1). The interrupts are controlled by IOC1.2 and IOC1.3 respectively. The flags are set in IOS1.5 and IOS1.4, respectively.

Caution must be used when examining the flags, as any access (including Compare and Jump on Bit) of IOS1 clears the whole byte, including the software timer flags. It is, therefore, recommended to write the byte to a temporary register before testing bits. The general enabling and disabling of the timer interrupts are controlled by the Interrupt Mask Register bit 0. In all cases, setting a bit enables a function, while clearing a bit disables it.

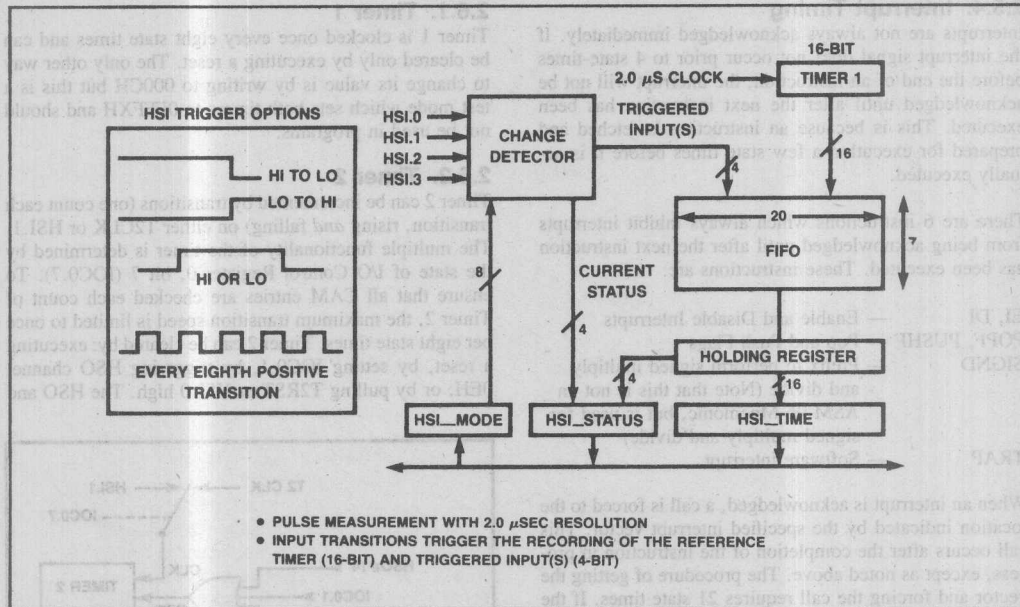


Figure 2-14. High Speed Input Unit

## 2.6.4. Timer Related Sections

The High Speed I/O unit is coupled to the timers in that the HSI records the value on Timer 1 when transitions occur and the HSO causes transitions to occur based on values of either Timer 1 or Timer 2. The Baud rate generator can use the T2CLK pin as input to its counter. A complete listing of the functions of IOS1, IOC0, and IOC1 are in section 2.13.

## 2.7. HIGH SPEED INPUTS

The High Speed Input Unit (HSI), can be used to record the time at which an event occurs with respect to Timer 1. There are 4 lines (HSI.0 through HSI.3) which can be used in this mode and up to a total of 8 events can be recorded. HSI.2 and HSI.3 share pins with HSO.4 and HSO.5. The I/O Control Registers (IOC0 and IOC1) are used to determine the functions of these pins. A block diagram of the HSI unit is shown in Figure 2-14.

### 2.7.1. HSI Modes

There are 4 possible modes of operation for each of the HSI. The HSI mode register is used to control which pins will look for what type of events. The 8-bit register is set up as shown in Figure 2-15. High and low levels each need to be held for at least 1 state time to ensure proper operation. The maximum input speed is 1 event every 8 state times except when the 8 transition mode is used, in which case it is 1 transition per state time.

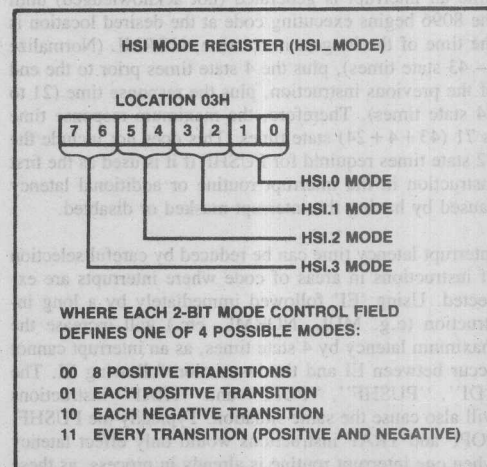
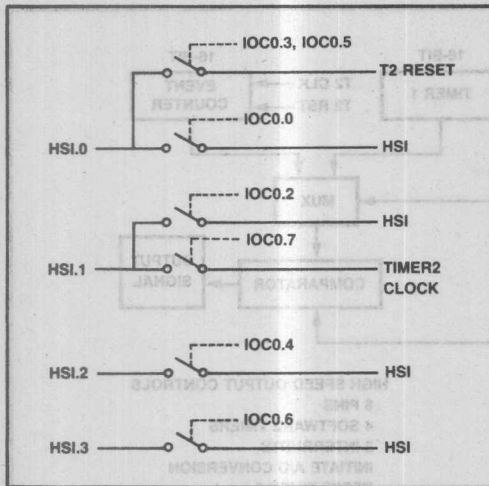


Figure 2-15. HSI Mode Register Diagram

The HSI lines can be individually enabled and disabled using bits in IOC0, at location 0015H. Figure 2-16 shows the bit locations which control the HSI pins. If the pin is disabled, transitions will not be entered in the FIFO.





**Figure 2-16. IOC0 Control of HSI Pin Functions**

## 2.7.2. HSI FIFO

When an HSI event occurs, a 7x20 FIFO stores the 16 bits of Timer 1 and the 4 bits indicating the state of the 4 HSI lines at the time the status is read. It can take up to 8 state times for this information to reach the holding register. When the FIFO is full, one additional event can be stored by considering the holding register part of the FIFO. If the FIFO and holding register are full any additional events will not be recorded.

## 2.7.3. HSI Interrupts

Interrupts can be generated from the HSI unit in one of two ways, determined by IOC1.7. If the bit is a 0, then an interrupt will be generated every time a value is loaded into the holding register. If it is a 1, an interrupt will only be generated when the FIFO, (independent of the holding register), has six entries in it. Since all interrupts are rising edge triggered, if IOC1.7=1, the processor will not be re-interrupted until the FIFO first contains 5 or less records, then contains six or more. Interrupts can also be generated by pin HSI.0, which has its own interrupt vector.

## 2.7.4. HSI Status

Bits 6 and 7 of the I/O Status register 1 (IOS1) indicate the status of the HSI FIFO. If bit 6 is a 1, the FIFO contains at least six entries. If bit 7 is a 1, the FIFO contains at least 1 entry and the holding register has been loaded. The FIFO may be read after verifying that it contains valid data. Caution must be used when reading or testing bits in IOS1, as this action clears the entire byte, including the software and hardware timer overflow flags. It is best to store the byte and then test the stored value. See Section 3.7.2.

Reading the HSI is done in two steps. First, the HSI Status

register is read to obtain the current state of the HSI pins and which pins had changed at the recorded time. The format of the HSI \_ STATUS Register is shown in Figure 2-17. Second, the HSI Time register is read. Reading the Time register unloads one word of the FIFO, so if the Time register is read before the Status register, the information in the Status register will be lost. The HSI Status register is at location 06H and the HSI Time registers are in locations 04H and 05H.

If the HSI \_ TIME and Status register are read without the holding register being loaded, the values read will be undeterminate.

It should be noted that many of the Status register conditions are changed by a reset, see section 2.15.2. A complete listing of the functions of IOS0, IOS1, and IOC1 can be found in section 2.13.

## 2.8. HIGH SPEED OUTPUTS

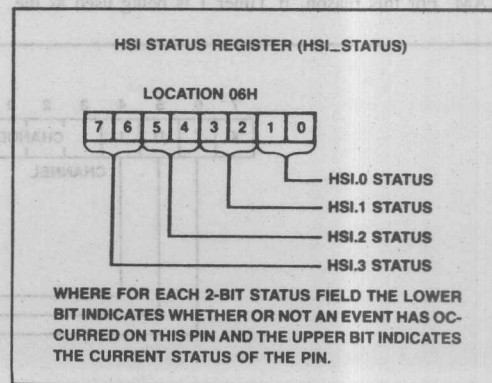
The High Speed Output unit (HSO) is used to trigger events at specific times with minimal CPU overhead. These events include: starting an A to D conversion, resetting Timer 2, setting 4 software flags, and switching up to 6 output lines. Interrupts can be generated whenever one of these events is triggered. Up to 8 events can be pending at any one time.

### 2.8.1. HSIO Shared Pins

Two of the 6 output lines (HSO.0 through HSO.5) are shared with the High Speed Input (HSI) lines. HSO.4 and HSO.5 are shared with HSI.2 and HSI.3, respectively. Bits 4 and 6 of the I/O Control Register 1 (IOC1) are used to enable HSO.4 and HSO.5 as outputs.

### 2.8.2. HSIO CAM

A block diagram of the HSO unit is shown in Figure 2-18. The Content Addressable Memory (CAM) file is the center of control. One CAM register is compared with a time value every state time. Therefore, it takes 8 state times to compare all CAM registers with a timer.



**Figure 2-17. HSI Status Register Diagram**

## ARCHITECTURAL OVERVIEW

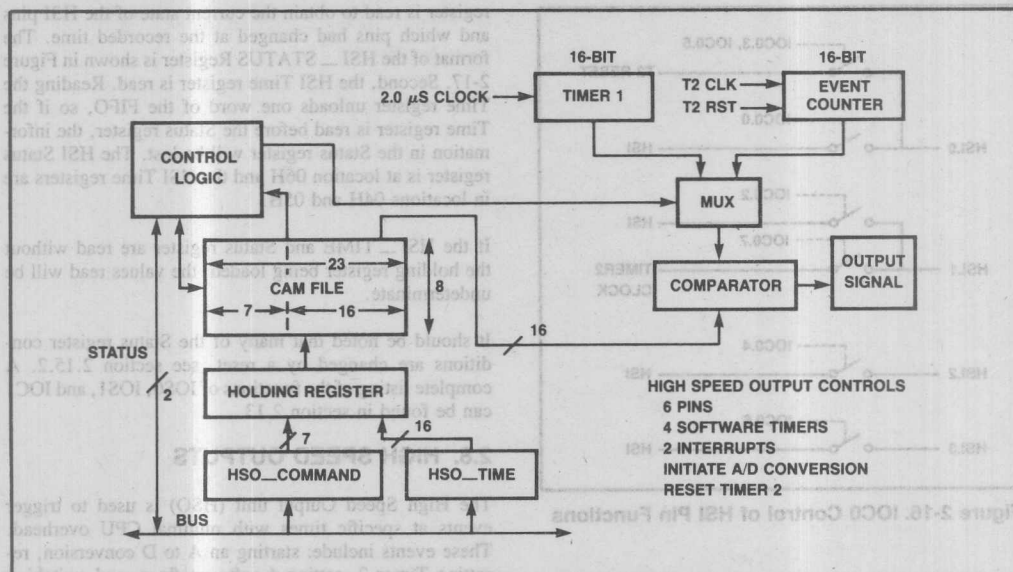


Figure 2-18. High Speed Output Unit

Each CAM register is 23 bits wide. Sixteen bits specify the time at which the action is to be carried out and 7 bits specify both the nature of the action and whether Timer 1 or Timer 2 is the reference. The format of the command to the HSO unit is shown in Figure 2-19.

To enter a command into the CAM file, write the 7-bit "Command Tag" into location 0006H followed by the time at which the action is to be carried out into word address 0004H. Writing the time value loads the HSO Holding Register with both the time and the last written command tag. The command does not actually enter the CAM file until an empty CAM register becomes available. It can take up to 8 state times for a command to enter the CAM. For this reason, if Timer 1 is being used as the

reference, the minimum time that can be loaded is Timer 1 + 1. A similar restriction applies if Timer 2 is used as the reference.

Care must be taken when writing the command tag for the HSO. If an interrupt occurs during the time between writing the command tag and loading the time value, and the interrupt service routine writes to the HSO time register, the command tag used in the interrupt routine will be written to the CAM at both the time specified by the interrupt routine and the time specified by the main program. The command tag from the main program will not be executed. One way of avoiding this problem would be to disable interrupts when writing commands and times to the HSO unit. See also Section 3.7.3.

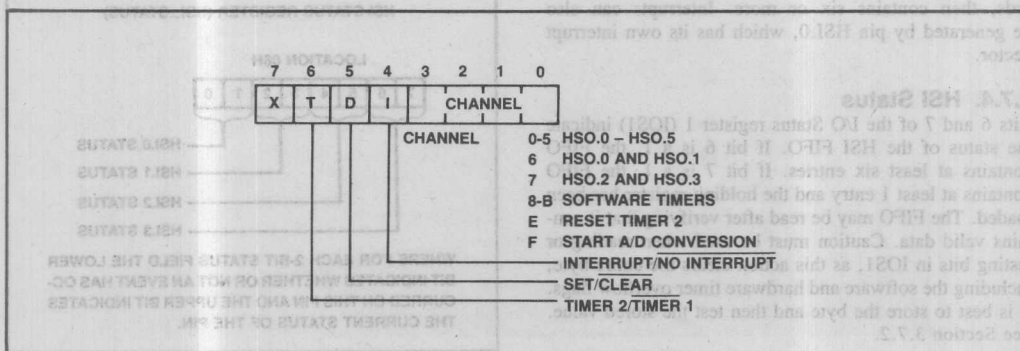


Figure 2-19. HSO Command Tag Format

## 2.8.3. HSO Status

Before writing to the HSO, it is desirable to ensure that the Holding Register is empty. If it is not, writing to the HSO will overwrite the value in the Holding Register. I/O Status Register 0 (IOS0) bits 6 and 7 indicate the status of the HSO unit. This register is described in section 2.13.4. If IOS0.6 equals 0, the holding register is empty and at least one CAM register is empty. If IOS0.7 equals 0, the holding register is empty.

One location in the CAM file is checked each state-time. Thus, it takes 8 state-times for the Holding Register to have had access to all 8 CAM registers. Similarly, it takes 8 state-times for the comparator to have had access to all 8 CAM registers. This defines the time-resolution of the HSO unit to be 8 state-times (2.0  $\mu$ sec, if the oscillator frequency is 12 MHz). Note that the comparator does not look at the holding register, so instructions in the holding register do not execute.

## 2.8.4. Clearing The HSO

All 8 CAM locations of the HSO are compared before any action is taken. This allows a pending external event to be cancelled by simply writing the opposite event to the CAM. However, once an entry is placed in the CAM, it cannot be removed until either the specified timer matches the written value or the chip is reset. Internal events are not synchronized to Timer 1, and therefore cannot be cleared. This includes events on HSO channels 8 through F and all interrupts. Since interrupts are not synchronized it is possible to have multiple interrupts at the same time value.

## 2.8.5. Using Timer 2 With The HSO

Timer 1 is incremented only once every 8 state-times. When it is being used as the reference timer for an HSO action, the comparator has a chance to look at all 8 CAM registers before Timer 1 changes its value. Following the same reasoning, Timer 2 has been synchronized to allow it to change at a maximum rate of once per 8 state-times. Timer 2 increments on both edges of the input signal.

When using Timer 2 as the HSO reference, caution must be taken that Timer 2 is not reset prior to the highest value for a Timer 2 match in the CAM. This is because the HSO CAM will hold an event pending until a time match occurs, if that match is to a time value on Timer 2 which is never reached, the event will remain pending in the CAM until the part is reset.

Additional caution must be used when Timer 2 is being reset using the HSO unit, since resetting Timer 2 using the HSO is an internal event and can therefore happen at any time within the eight-state-time window. For this reason, any events scheduled to occur at the same time as a Timer 2 reset should be logged into the CAM with a Timer 2 value of zero. When using this method to make a programmable modulo counter, the count will stay at the maximum Timer 2 value only until the Reset T2 command is recognized. The count will stay at zero for the transition which would have changed the count from 'N' to zero, and then change to a one on the next transition.

## 2.8.6. Software Timers

The HSO can be programmed to generate interrupts at preset times. Up to four such "Software Timers" can be in operation at a time. As each preprogrammed time is reached, the HSO unit sets a Software Timer Flag. If the interrupt bit in the command tag was set then a Software Timer Interrupt will also be generated. The interrupt service routine can then examine I/O Status register 1 (IOS1) to determine which software timer expired and caused the interrupt. When the HSO resets Timer 2 or starts an A to D conversion, it can also be programmed to generate a software timer interrupt but there is no flag to indicate that this has occurred. See also Section 3.7.4.

If more than one software timer interrupt occurs in the same time frame it is possible that multiple software timer interrupts will be generated.

Each read or test of any bit in IOS1 will clear the whole byte. Be certain to save the byte before testing it unless you are only concerned with 1 bit. See also Section 3.2.2.

A complete listing of the functions of IOS0, IOS1, and IOC1 can be found in section 2.13. The Timers are described in section 2.6 and the HSI is described in section 2.7.

## 2.9. ANALOG INPUTS

The A to D converter on the 8096 provides a 10-bit result on one of 8 input channels. Conversion is done using successive approximation with a result equal to the ratio of the input voltage divided by the analog supply voltage. If the ratio is 1.00, then the result will be all ones. The A/D converter is available on the 8097, 8397, 8095 and 8395 members of the MCS<sup>®</sup>-96 family.

### 2.9.1. A/D Accuracy

Each conversion requires 168 state-times (42 $\mu$ s at 12 MHz) independent of the accuracy desired or value of input voltage. The input voltage must be in the range of 0 to VREF, the analog reference and supply voltage. For proper operation, VREF (the reference voltage and analog power supply) must be held at VCC  $\pm$  0.3V with VREF = 5.0  $\pm$  0.5V. The A/D result is calculated from the formula:

$$1023 \times (\text{input voltage} - \text{ANGND}) / (\text{VREF} - \text{ANGND})$$

It can be seen from this formula that changes in VREF or ANGND effect the output of the converter. This can be advantageous if a ratiometric sensor is used since these sensors have an output that can be measured as a proportion of VREF.

If high absolute accuracy is needed it may be desirable to use a separate power supply, or power traces, to operate the A/D converter. There is no sample and hold circuit internal to the chip, so the input voltage must be held constant for the entire 168 state times. Examples of connecting the A/D converter to various devices are given in section 4.3.



## ARCHITECTURAL OVERVIEW

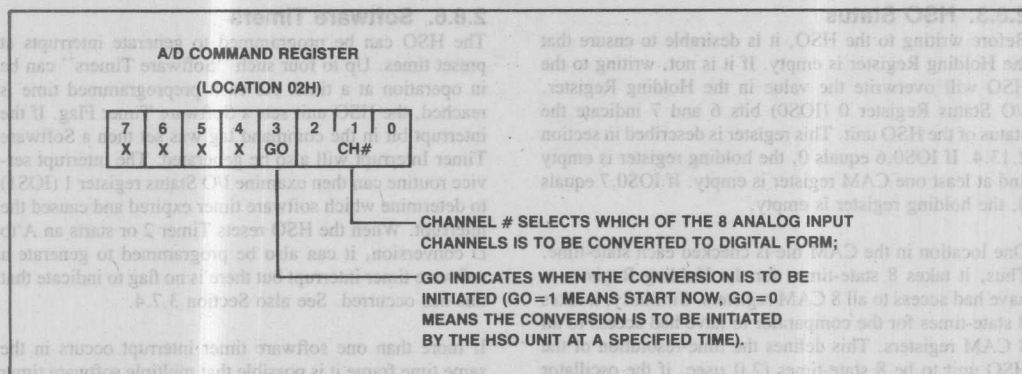


Figure 2-20. A/D Command Register

### 2.9.2. A/D Commands

Analog signals can be sampled by any one of the 8 analog input pins (ACH0 through ACH7) which are shared with Port 0. ACH7 can also be used as an external interrupt if IOC1.1 is set (see section 2.5). The A/D Command Register, at location 02H, selects which channel is to be converted and whether the conversion should start immediately or when the HSO (Channel #0FH) triggers it. A to D commands are formatted as shown in Figure 2-20.

The command register is double buffered so it is possible to write a command to start a conversion triggered by the HSO while one is still in progress. Care must be taken when this is done since if a new conversion is started while one is already in progress, the conversion in progress is cancelled and the new one is started. When a conversion is started, the result register is cleared. For this reason the result register must be read before a new conversion is started or data will be lost.

### 2.9.3. A/D Results

Results of the analog conversions are read from the A/D

Result Register at locations 02H and 03H. Although these addresses are on a word boundary, they must be read as individual bytes. Information in the A/D Result register is formatted as shown in Figure 2-21. Note that the status bit may not be set until 8 state times after the go command. Information on using the HSO is in section 2.8.

### 2.10. PULSE WIDTH MODULATION OUTPUT (D/A)

Digital to analog conversion can be done with the pulse width modulation output; a block diagram of the circuit is shown in Figure 2-22. The 8-bit counter is incremented every state time. When it equals 0, the PWM output is set to a one. When the counter matches the value in the PWM register, the output is switched low. When the counter overflows, the output is once again switched high. A typical output waveform is shown in Figure 2-23. Note that when the PWM register equals 00, the output is always low.

The output waveform is a variable duty cycle pulse which repeats every 256 state times (64  $\mu$ S at 12MHz). Changes

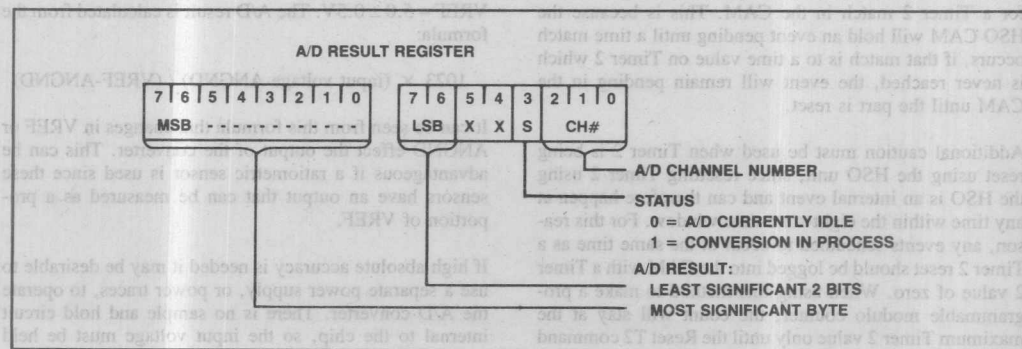


Figure 2-21. A/D Result Register

## ARCHITECTURAL OVERVIEW

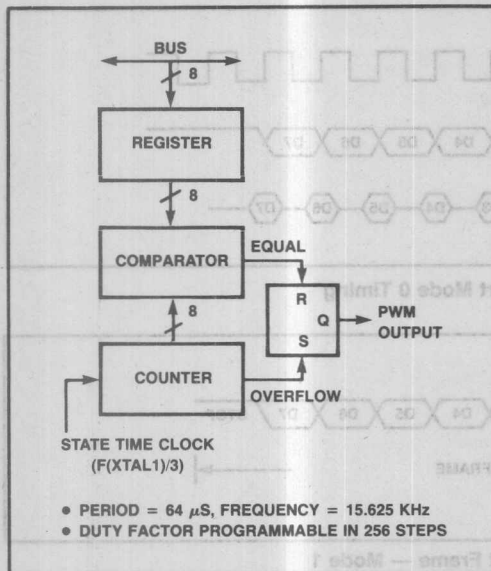


Figure 2-22. Pulse Width Modulated (D/A) Output

in the duty cycle are made by writing to the PWM register at location 17H. There are several types of motors which require a PWM waveform for most efficient operation. Additionally, if this waveform is integrated it will produce a DC level which can be changed in 256 steps by varying the duty cycle.

Details about the hardware required for smooth, accurate D/A conversion can be found in section 4.3.2. Typically,

some form of buffer and integrator are needed to obtain the most usefulness from this feature.

The PWM output shares a pin with Port 2, pin 5 so that these two features cannot be used at the same time. IOC1.0 equal to 1 selects the PWM function instead of the standard port function. More information on IOC1 is in section 2.13.3.

## 2.11. SERIAL PORT

The serial port is compatible with the MCS-51 serial port. It is full duplex, meaning it can transmit and receive simultaneously. It is also receive-buffered, meaning it can commence reception of a second byte before a previously received byte has been read from the receive register. The serial port registers (SBUF) are both accessed at location 07H. A write to this location accesses the transmit register, and a read accesses a physically separate receive register.

The serial port can operate in 4 modes (explained below). Selection of these modes is done through the Serial Port Status/Control register at location 11H, shown in Figure 2-27.

### 2.11.1. Serial Port Modes

#### MODE 0

Mode 0 is a shift register mode. The 8096 outputs a train of 8 shift pulses to an external shift register to clock 8 bits of data into or out of the register from or to the 8096. Serial data enters and exits the 8096 through RXD. TXD outputs the shift clock. 8 bits are transmitted or received, LSB first. A timing diagram of this mode is shown in Figure 2-24. This mode is useful as an I/O expander in which application external shift registers can be used as additional parallel I/O ports. An example of using the port in this mode is given in section 4.5.

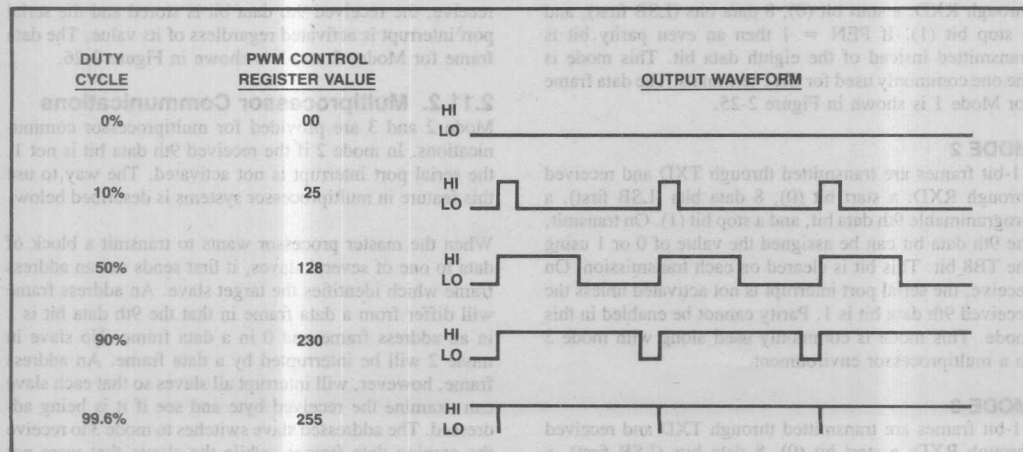


Figure 2-23. Typical PWM Outputs

## ARCHITECTURAL OVERVIEW

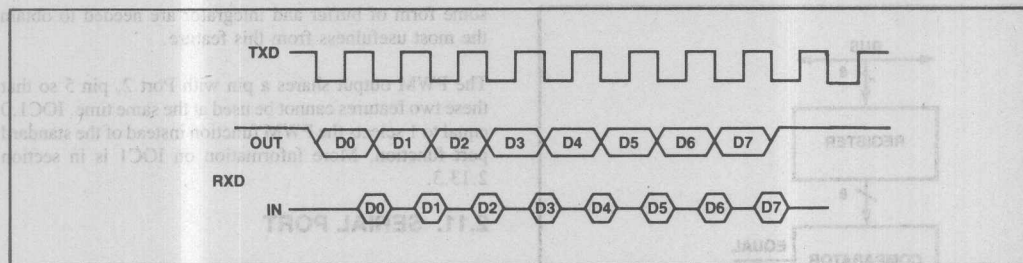


Figure 2-24. Serial Port Mode 0 Timing

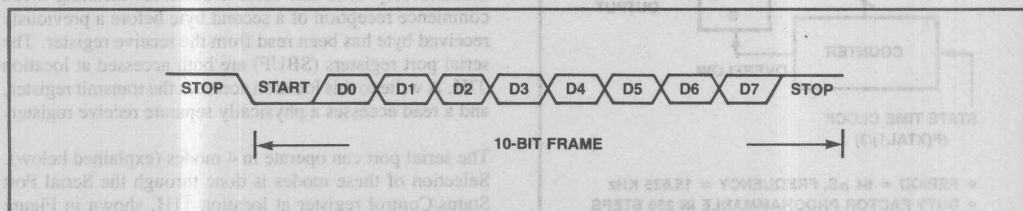


Figure 2-25. Serial Port Frame — Mode 1

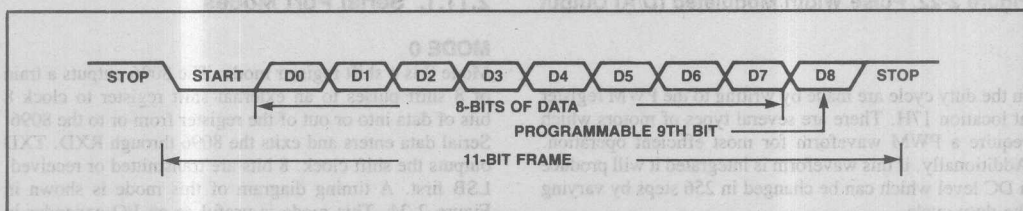


Figure 2-26. Serial Port Frame Modes 2 and 3

### MODE 1

10-bit frames are transmitted through TXD, and received through RXD: a start bit (0), 8 data bits (LSB first), and a stop bit (1). If  $PEN = 1$  then an even parity bit is transmitted instead of the eighth data bit. This mode is the one commonly used for CRT terminals. The data frame for Mode 1 is shown in Figure 2-25.

### MODE 2

11-bit frames are transmitted through TXD and received through RXD: a start bit (0), 8 data bits (LSB first), a programmable 9th data bit, and a stop bit (1). On transmit, the 9th data bit can be assigned the value of 0 or 1 using the TB8 bit. This bit is cleared on each transmission. On receive, the serial port interrupt is not activated unless the received 9th data bit is 1. Parity cannot be enabled in this mode. This mode is commonly used along with mode 3 in a multiprocessor environment.

### MODE 3

11-bit frames are transmitted through TXD and received through RXD: a start bit (0), 8 data bits (LSB first), a programmable 9th data bit, and a stop bit (1). On transmit,

the 9th data bit can be assigned the value of 0 or 1 using the TB8 bit. If  $PEN = 1$  the 9th bit will be parity. On receive, the received 9th data bit is stored and the serial port interrupt is activated regardless of its value. The data frame for Modes 2 and 3 is shown in Figure 2-26.

### 2.11.2. Multiprocessor Communications

Mode 2 and 3 are provided for multiprocessor communications. In mode 2 if the received 9th data bit is not 1, the serial port interrupt is not activated. The way to use this feature in multiprocessor systems is described below.

When the master processor wants to transmit a block of data to one of several slaves, it first sends out an address frame which identifies the target slave. An address frame will differ from a data frame in that the 9th data bit is 1 in an address frame and 0 in a data frame. No slave in mode 2 will be interrupted by a data frame. An address frame, however, will interrupt all slaves so that each slave can examine the received byte and see if it is being addressed. The addressed slave switches to mode 3 to receive the coming data frames, while the slaves that were not addressed stay in mode 2 and go on about their business.

## 2.11.3. Controlling the Serial Port

Control of the Serial Port is done through the Serial Port Control/Status register. The format for the control word is shown in Figure 2-27. Note that reads access only part of the byte, as do writes, and that TB8 is cleared after each byte is transmitted.

In Mode 0, if REN=0, writing to SBUF will start a transmission. Causing a rising edge on REN, or clearing RI with REN=1, will start a reception. Setting REN=0 will stop a reception in progress, and inhibit further receptions. To avoid a partial or complete undesired reception, REN must be set to zero before clearing RI. This can be handled in an interrupt environment by using software flags, or in a straight-line code environment by using the Interrupt Pending register to signal the completion of a receive. In any mode, it is necessary to set IOC1.5 to a 1 to enable the TXD pin. Some examples of the software involved in using the serial port can be found in section 3.8. More information on IOC1 is in section 2.13.3.

## 2.11.4. Determining Baud Rates

Baud rates in all modes are determined by the contents of a 16-bit register at location 000EH. This register must be loaded sequentially with 2 bytes (least significant byte first). The MSB of this register selects one of two sources for the input frequency to the baud rate generator. If it is a 1, the frequency on the XTAL1 pin is selected, if not, the external frequency from the T2CLK pin is used. It should be noted that the maximum speed of T2CLK is one transition every 2 state times, with a minimum period of 16 XTAL1 cycles.

The unsigned integer represented by the lower 15 bits of the baud rate register defines a number B, where B has a maximum value of 32767. The baud rate for the four serial modes using either XTAL1 or T2CLK as the clock source is given by:

Using XTAL1:

$$\text{Mode 0: Baud Rate} = \frac{\text{XTAL1 frequency}}{4*(B+1)}; B \neq 0$$

$$\text{Others: Baud Rate} = \frac{\text{XTAL1 frequency}}{64*(B+1)}$$

Using T2CLK:

$$\text{Mode 0: Baud Rate} = \frac{\text{T2CLK frequency}}{B}; B \neq 0$$

$$\text{Others: Baud Rate} = \frac{\text{T2CLK frequency}}{16*B}; B \neq 0$$

Note that B cannot equal 0, except when using XTAL1 in other than mode 0.

Common baud rate values, using XTAL1 at 12MHz, are shown below.

Baud Rate	Baud Register Mode 0	Value Others
9600	8137H	8013H
4800	8270H	8026H
2400	84E1H	804DH
1200	89C3H	809BH
300	A70FH	8270H

The maximum asynchronous baud rate is 187.5 Kbaud, with a 12MHz clock.

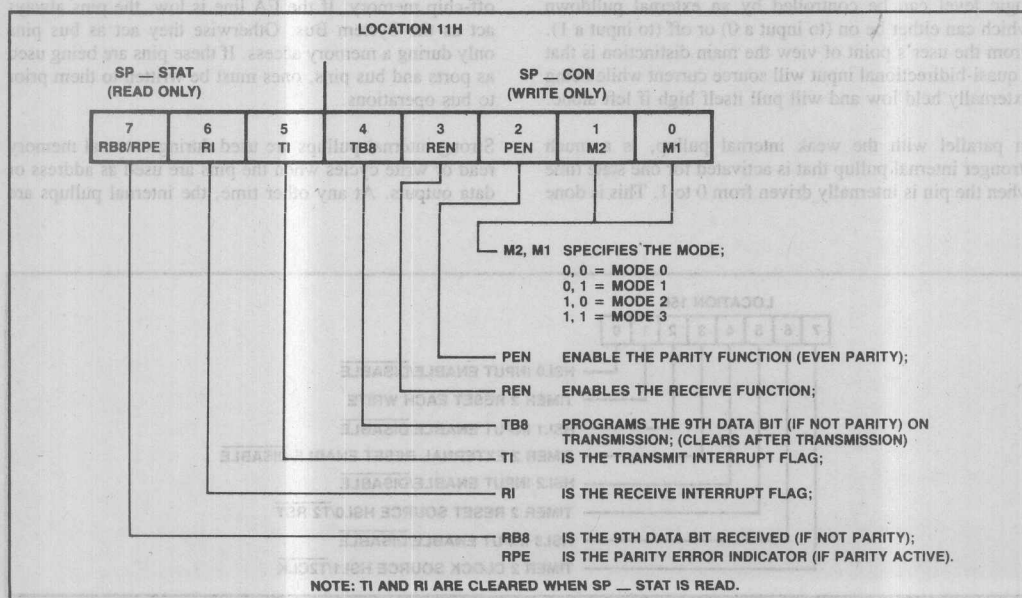


Figure 2-27. Serial Port Control/Status Register



## 2.12. I/O PORTS 0, 1, 2, 3, AND 4

There are five 8-bit I/O ports on the 8096. Some of these ports are input only, some output only, some bidirectional and some have alternate functions. Input ports connect to the internal bus through an input buffer. Output ports connect through an output buffer to an internal register that holds the output bits. Bidirectional ports consist of an internal register, an output buffer, and an input buffer.

When an instruction accesses a bidirectional port as a source register, the question often arises as to whether the value that is brought into the CPU comes from the internal port register or from the port pins through the input buffer. In the 8096, the value always comes from the port pins, never from the internal register.

### 2.12.1. Port 0

Port 0 is an input only port which shares its pins with the analog inputs to the A/D Converter. One can read Port 0 digitally, and/or, by writing the appropriate control bits to the A/D Command Register, select one of the lines of this port to be the input to the A/D Converter. While a conversion is in process, the impedance of the selected line is lower than normal. See the data sheet for the specific values.

### 2.12.2. Port 1

Port 1 is a quasi-bidirectional I/O port. "Quasi-bidirectional" means the port pin has a weak internal pullup that is always active and an internal pulldown which can either be on (to output a 0) or off (to output a 1). If the internal pulldown is left off (by writing a 1 to the pin), the pin's logic level can be controlled by an external pulldown which can either be on (to input a 0) or off (to input a 1). From the user's point of view the main distinction is that a quasi-bidirectional input will source current while being externally held low and will pull itself high if left alone.

In parallel with the weak internal pullup, is a much stronger internal pullup that is activated for one state time when the pin is internally driven from 0 to 1. This is done

to speed up the 0-to-1 transition time. See also Sections 3.7.1 and 4.2.2.

### 2.12.3. Port 2

Port 2 is a multi-functional port. Six of its pins are shared with other functions in the 8096, as shown below.

Port	Function	Alternate Function	Controlled by
P2.0	output	TXD (serial port transmit)	IOC1.5
P2.1	input	RXD (serial port receive)	N/A
P2.2	input	EXTINT (external interrupt)	IOC1.1
P2.3	input	T2CLK (Timer 2 input)	IOC0.7
P2.4	input	T2RST (Timer 2 reset)	IOC0.5
P2.5	output	PWM (pulse-width modulation)	IOC1.0
P2.6	quasi-bidirectional		
P2.7	quasi-bidirectional		

### 2.12.4. Ports 3 and 4

Ports 3 and 4 have two functions. They are either bidirectional ports with open-drain outputs or System Bus pins which the memory controller uses when it is accessing off-chip memory. If the EA line is low, the pins always act as the System Bus. Otherwise they act as bus pins only during a memory access. If these pins are being used as ports and bus pins, ones must be written to them prior to bus operations.

Strong internal pullups are used during external memory read or write cycles when the pins are used as address or data outputs. At any other time, the internal pullups are

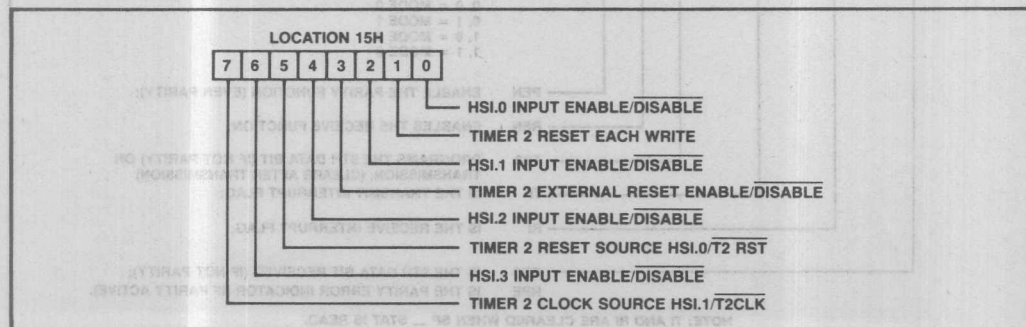


Figure 2-28. I/O Control Register 0 (IOC0)

disabled. The port pins and their system bus functions are shown below:

Port Pin	System Bus Function
P3.0	AD0
P3.1	AD1
P3.2	AD2
P3.3	AD3
P3.4	AD4
P3.5	AD5
P3.6	AD6
P3.7	AD7
P4.0	AD8
P4.1	AD9
P4.2	AD10
P4.3	AD11
P4.4	AD12
P4.5	AD13
P4.6	AD14
P4.7	AD15

## 2.13. STATUS AND CONTROL REGISTERS

### 2.13.1. I/O Control Registers

There are two I/O Control registers, IOC0 and IOC1. IOC0 controls Timer 2 and the HSI lines. IOC1 controls some pin functions, interrupt sources and 2 HSO pins.

Whenever input lines are switched between two sources, or enabled, it is possible to generate transitions on these lines. This could cause problems with respect to edge sensitive lines such as the HSI lines, Interrupt line, and Timer 2 control lines.

### 2.13.2. I/O Control Register 0 (IOC0)

IOC0 is located at 0015H. The four HSI lines can be enabled or disabled to the HSI unit by setting or clearing bits in IOC0. Timer 2 functions including clock and reset sources are also determined by IOC0. The control bit locations are shown in Figure 2-28.

### 2.13.3. I/O Control Register 1 (IOC1)

IOC1 is used to select some pin functions and enable or disable some interrupt sources. Its location is 0016H. Port pin P2.5 can be selected to be the PWM output instead of a standard output. The external interrupt source can be selected to be either EXTINT (same pin as P2.2) or Analog Channel 7 (ACH7, same pin as P0.7). Timer 1 and Timer 2 overflow interrupts can be individually enabled or disabled. The HSI interrupt can be selected to activate either when there is 1 FIFO entry or 7. Port pin P2.0 can be selected to be the TXD output. HSO.4 and HSO.5 can be enabled or disabled to the HSO unit. More information on interrupts is available in section 2.5. The positions of the IOC1 control bits are shown in Figure 2-29.

### 2.13.4. I/O Status Register 0 (IOS0)

There are two I/O Status registers, IOS0 and IOS1. IOS0, located at 0015H, holds the current status of the HSO lines and CAM. The status bits of IOS0 are shown in Figure 2-30.

### 2.13.5. I/O Status Register 1 (IOS1)

IOS1 is located at 0016H. It contains status bits for the timers and the HSI. Every access of this register clears all of the timer overflow and timer expired bits. It is, therefore, important to first store the byte in a temporary location before attempting to test any bit unless only one bit will ever be of importance to the program. The status bits of IOS1 are shown in Figure 2-31.

## 2.14. WATCHDOG TIMER (WDT)

This feature is provided as a means of graceful recovery from a software upset. Once the watchdog is initialized, if the software fails to reset the watchdog at least every 64K state times, a hardware reset will be initiated.

The watchdog is initialized by the first write of the clear WDT code to the WDT register. Once the watchdog is initialized it cannot be disabled by software. On reset the watchdog is not active.

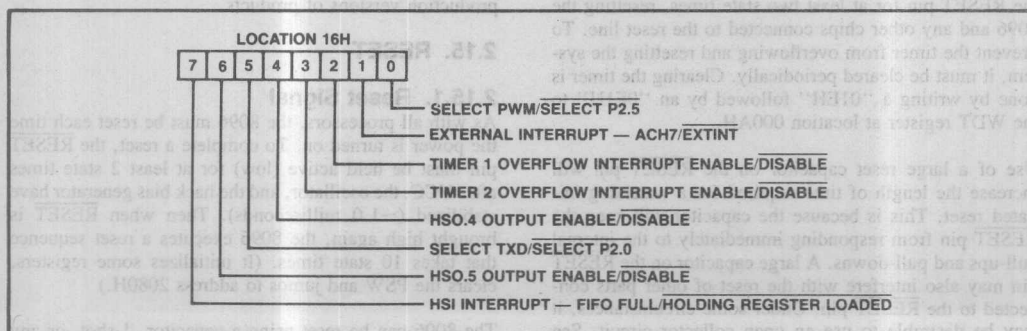


Figure 2-29. I/O Control Register 1 (IOC1)

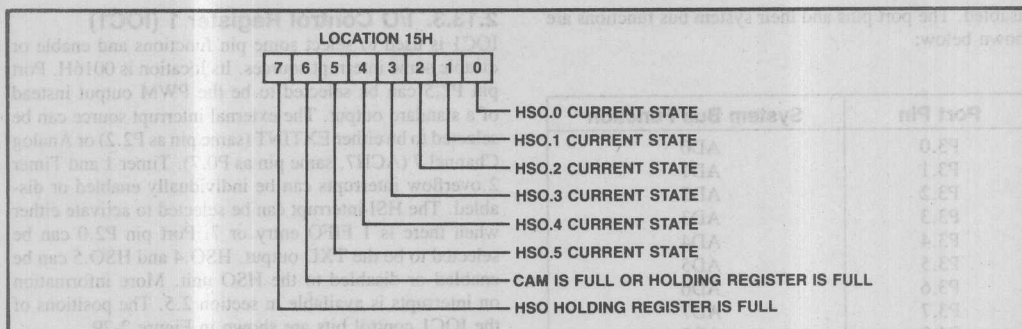


Figure 2-30. I/O Status Register 0 (IOS0)

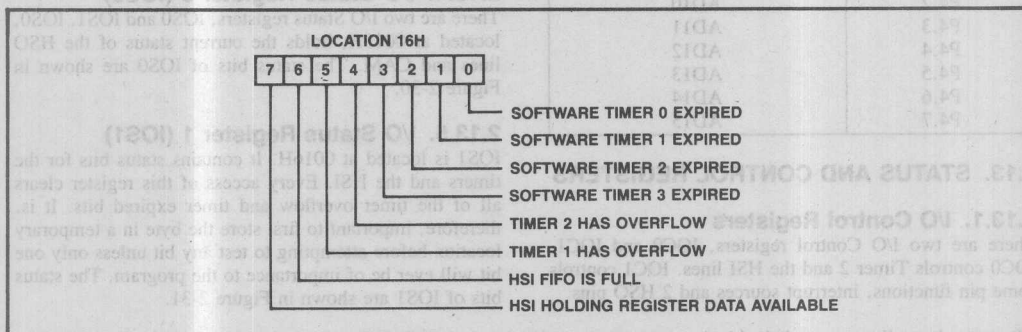


Figure 2-31. HSIO Status Register 1 (IOS1)

The software can be designed so that the watchdog times out if the program does not progress properly. The watchdog will also time-out if the software error was due to ESD (Electrostatic Discharge) or other hardware related problems. This prevents the controller from having a malfunction for longer than 16 mS if a 12 MHz oscillator is used.

The watchdog timer is a 16-bit counter which is incremented every state time. When it overflows it pulls down the **RESET** pin for at least two state times, resetting the 8096 and any other chips connected to the reset line. To prevent the timer from overflowing and resetting the system, it must be cleared periodically. Clearing the timer is done by writing a "01EH" followed by an "0E1H" to the WDT register at location 000AH.

Use of a large reset capacitor on the **RESET** pin will increase the length of time required for a watchdog initiated reset. This is because the capacitor will keep the **RESET** pin from responding immediately to the internal pull-ups and pull-downs. A large capacitor on the **RESET** pin may also interfere with the reset of other parts connected to the **RESET** pin. Under some circumstances, it may be desirable to use an open collector circuit. See section 4.4.

## 2.14.1. Disabling The Watchdog

The watchdog should be disabled by software not initializing it. If this is not possible, such as during program development, the watchdog can be disabled by holding the **RESET** pin at 2.0 to 2.5 volts. Voltages over 2.5 volts on the pin could quickly damage the part. Even at 2.5 volts, using this technique for other than debugging purposes is not recommended, as it may effect long term reliability. It is further recommended that any part used in this way for more than several seconds, not be used in production versions of products.

## 2.15. RESET

### 2.15.1. Reset Signal

As with all processors, the 8096 must be reset each time the power is turned on. To complete a reset, the **RESET** pin must be held active (low) for at least 2 state times after VCC, the oscillator, and the back bias generator have stabilized (~1.0 milliseconds). Then when **RESET** is brought high again, the 8096 executes a reset sequence that takes 10 state times. (It initializes some registers, clears the PSW and jumps to address 2080H.)

The 8096 can be reset using a capacitor, 1-shot, or any other method capable of providing a pulse of at least 2



## ARCHITECTURAL OVERVIEW

state times longer than required for VCC and the oscillator to stabilize.

For best functionality, it is suggested that the reset pin be pulled low with an open collector device. In this way, several reset sources can be wire or'd together. Remember, the RESET pin itself can be a reset source (see section 2.14). Details of hardware suggestions for reset can be found in section 4.4.

### 2.15.2. Reset Status

The I/O lines and control of the 8096 will be in their reset state within 2 state times after reset is low, with VCC and the oscillator stabilized. Prior to that time, the status of the I/O lines is indeterminate. After the 10 state time reset sequence, the Special Function Registers will be set as follows:

SFR	reset value
Port 1	11111111B
Port 2	110XXXX1B
Port 3	11111111B
Port 4	11111111B
PWM Control	00H
Serial Port (Transmit)	undefined
Serial Port (Receive)	undefined
Baud Rate Register	undefined
Serial Control/Status	undefined
A/D Command	undefined
A/D Result	undefined
Interrupt Pending	undefined
Interrupt Mask	00000000B
Timer 1	0000H
Timer 2	0000H
Watchdog Timer	0000H
HSI Mode	11111111B
HSI Status	undefined
IOS0	00000000B
IOS1	00000000B
IOC0	X0X0X0X0B
IOC1	X0X0XXX1B

Other conditions following a reset are:

Register	reset value
HSI FIFO	empty
HSO CAM	empty
HSO lines	000000B
PSW	0000H
Stack Pointer	undefined
Program Counter	2080H
RD	high
WR	high
ALE	low
BHE	low
INST	high

### NMI

A low to high transition causes a vector to external memory location 0000H. Reserved for use in Intel Development systems.

It is important to note that the Stack Pointer and Interrupt Pending Register are undefined, and need to be initialized in software. The Interrupts are disabled by both the mask register and PSW.9 after a reset.

### 2.15.3. Reset Sync Mode

The RESET line can be used to start the 8096 at an exact state time to provide for synchronization of test equipment and multiple chip systems. RESET is active low. To synchronize parts, RESET is brought high on the rising edge of XTAL1. Complete details on synchronizing parts can be found in section 4.1.5.

It is very possible that parts which start in sync may not stay that way. The best example of this would be when a "jump on I/O bit" is being used to hold the processor in a loop. If the line changes during the time it is being tested, one processor may see it as a one, while the other sees it as a zero. The result is that one processor will do an extra loop, thus putting it several states out of sync with the other.

## 2.16. PIN DESCRIPTION

### VCC

Main supply voltage (5V).

### VSS

Digital circuit ground (0V). There are two VSS pins, both must be tied to ground.

### VPD

RAM standby supply voltage (5V). This voltage must be present during normal operation. See section 2.4.2 and 4.

### VREF

Reference voltage and power supply for the analog portion of the A/D converter. Nominally at 5 volts. See section 2.9.1 and 4.

### ANGND

Reference ground for the A/D converter. Should be held at nominally the same potential as VSS. See section 2.9.1 and 4.

### VBB

Substrate voltage from the on-chip back-bias generator. This pin should be connected to ANGND through a 0.01 uf capacitor (and not connected to anything else). The capacitor is not required if the A/D converter is not being used.

### XTAL1

Input of the oscillator inverter and input to the internal clock generator. See sections 2.2 and 4.



## XTAL2

Output of the oscillator inverter. See section 2.2. A low to high transition causes a vector to external memory location 0000H. Reserved for use in Intel Development systems.

## CLKOUT

Output of the internal clock generator. The frequency of CLKOUT is 1/2 the oscillator frequency. It has a 33% duty cycle. CLKOUT can drive one TTL input. See section 2.2. The interrupt is disabled by both the register and PSW 9 after a read.

## RESET

Reset input to the chip, also output to other circuits. Input low for at least 2 state times to reset the chip. RESET has a strong internal pullup. See section 2.15 and 4.1. In state and multiple chip systems, RESET is active low. To enable the processor, RESET is brought high on the main bus.

## TEST

Input low enables a factory test mode. The user should tie this pin to VCC for normal operation. See section 2.1. It is found in some devices.

## NMI

A low to high transition causes a vector to external memory location 0000H. Reserved for use in Intel Development systems. In a loop, if the time changes during the test, one processor may see it as a one, while the other sees it as a zero. The result is that one processor reads the address is valid during an external read indicates the read is an instruction fetch. See section 2.3.6 and 4.6.

## INST

Output high while the address is valid during an external read indicates the read is an instruction fetch. See section 2.3.6 and 4.6.

## EA

Input for memory select (External Access).  $\overline{EA} = 1$  causes memory accesses to locations 2000H through 3FFFFH to be directed to on-chip ROM.  $\overline{EA} = 0$  causes accesses to these locations to be directed to off-chip memory. EA has an internal pulldown, so it goes to 0 unless driven to 1. See section 2.3.3. It must be tied to ground.

## ALE

Address Latch Enable output. ALE is activated only during external memory accesses. It is used to latch the address from the multiplexed address/data bus. See section 2.3.5 and 4.6.

## RD

Read signal output to external memory.  $\overline{RD}$  is activated only during external memory reads. See section 2.3.5 and 4.6.

## WR

Write signal output to external memory.  $\overline{WR}$  is activated only during external memory writes. See section 2.3.6 and 4.6. For best functionality, it is suggested that the pin be pulled low with an open collector device. In this way, several read sources can be wire and together. Remember, the RESET pin itself can be a read source (see section 2.1.5.2. Reset Status).

## BHE

Bus High Enable signal output to external memory.  $\overline{BHE}$  (0/1) selects/deselects the bank of memory that is connected to the high byte of the data bus. See section 2.3.5 and 4.6. The RD lines and control of the 8096 will be in their reset state within 2 state times after reset is low with VCC and the oscillator stabilized. Prior to that time, the RD lines and control of the 8096 will be in their reset state.

## READY

The READY input is used to lengthen external memory bus cycles up to the time specified in the data sheet. It has a weak internal pullup. See section 2.3.6 and 4.6.

## HSI

High impedance inputs to HSI Unit. Four HSI pins are available: HSI.0, HSI.1, HSI.2, and HSI.3. Two of them (HSI.2 and HSI.3) are shared with the HSO Unit. See section 2.7.

## HSO

Outputs from HSO Unit. Six HSO pins are available: HSO.0, HSO.1, HSO.2, HSO.3, HSO.4, and HSO.5. Two of them (HSO.4 and HSO.5) are shared with the HSI Unit. All HSO pins are capable of driving one TTL input. See section 2.8.

## PORT 0/ANALOG CHANNEL

High impedance input-only port. These pins can be used as digital inputs and/or as analog inputs to the on-chip A/D converter. See sections 2.9 and 2.12.1.

## PORT 1

Quasi-bidirectional I/O port. All pins of P1 are capable of driving one LS TTL input. See section 2.12.2.

## PORT 2

Multi-functional port. Six of its pins are shared with other functions in the 8096, as shown below.

Port	Function	Alternate Function	Reference section
P2.0	output	TXD (serial port transmit)	2.11.3
P2.1	input	RXD (serial port receive)	2.11.3
P2.2	input	EXTINT (external interrupt)	2.5
P2.3	input	T2CLK (Timer 2 input)	2.6.2
P2.4	input	T2RST (Timer 2 reset)	2.6.2
P2.5	output	PWM (pulse-width modulator)	2.10
P2.6	quasi-bidirectional		
P2.7	quasi-bidirectional		

## ARCHITECTURAL OVERVIEW

The multi-functional inputs are high impedance. See section 2.12.3.

### PORTS 3 AND 4

8-bit bidirectional I/O ports. These pins are shared the multiplexed address/data bus when accessing external memory, with the Port 3 pins accessing the low byte and Port 4 pins accessing the high byte. They are open drain except when being used as system bus pins. See section 2.3.5.

### 2.17. PIN LIST

The following is a list of pins in alphabetical order. Where a pin has two names it has been listed under both names, except for the system bus pins, AD0-AD15, which are listed under Port 3 and Port 4.

Name	68-Pin	48-Pin
ACH0/P0.0	4	—
ACH1/P0.1	5	—
ACH2/P0.2	3	—
ACH3/P0.3	6	—
ACH4/P0.4	67	43
ACH5/P0.5	68	42
ACH6/P0.6	2	40
ACH7/P0.7	1	41
ALE	16	34
ANGND	66	44
BHE	37	15
CLKOUT	13	—
EA	7	39
EXTINT/P2.2	63	47
HSI.0	54	3
HSI.1	53	4
HSI.2/HSO.4	52	5
HSI.3/HSO.5	51	6
HSO.0	50	7
HSO.1	49	8
HSO.2	44	9
HSO.3	43	10
HSO.4/HSI.2	52	5
HSO.5/HSI.3	51	6
INST	15	—
NMI	7	—
PWM/P2.5	39	13
P0.0/ACH0	4	—
P0.1/ACH1	5	—
P0.2/ACH2	3	—
P0.3/ACH3	6	—
P0.4/ACH4	67	43
P0.5/ACH5	68	42
P0.6/ACH6	2	40
P0.7/ACH7	1	41
P1.0	59	—

Name	68-Pin	48-Pin
P1.1	58	—
P1.2	57	—
P1.3	56	—
P1.4	55	—
P1.5	48	—
P1.6	47	—
P1.7	46	—
P2.0/TXD	60	2
P2.1/RXD	61	1
P2.2/EXTINT	63	47
P2.3/T2CLK	34	—
P2.4/T2RST	36	—
P2.5/PWM	39	13
P2.6	45	—
P2.7	40	—
P3.0/AD0	18	32
P3.1/AD1	19	31
P3.2/AD2	20	30
P3.3/AD3	21	29
P3.4/AD4	22	28
P3.5/AD5	23	27
P3.6/AD6	24	26
P3.7/AD7	25	25
P4.0/AD8	26	24
P4.1/AD9	27	23
P4.2/AD10	28	22
P4.3/AD11	29	21
P4.4/AD12	30	20
P4.5/AD13	31	19
P4.6/AD14	32	18
P4.7/AD15	33	17
RD	17	33
READY	35	16
RESET	62	48
RXD/P2.1	61	1
TEST	14	—
TXD/P2.0	60	2
T2CLK/P2.3	34	—
T2RST/P2.4	36	—
VBB	41	12
VCC	9	38
VPD	64	46
VREF	65	45
VSS	10	11
VSS	42	37
WR	38	14
XTAL1	11	36
XTAL2	12	35

The Following pins are not bonded out in the 48-pin package:

P1.0 through P1.7, P0.0 through P0.3, P2.3, P2.4, P2.6, P2.7 CLKOUT, INST, NMI, TEST, T2CLK(P2.3), T2RST(P2.4).

## PORTS 3 AND 4

8-pin bidirectional I/O pins. These pins are shared the multiplexed address/data bus when accessing external memory, with the Port 3 pins accessing the low byte and Port 4 pins accessing the high byte. They are open drain except when being used as system bus pins. See section 5.12.3.

## 5.12. PIN LIST

The following is a list of pins in alphabetical order. Where a pin has two names it has been listed under both names, except for the system bus pins, ADO-AID15, which are listed under Port 3 and Port 4.

Name	66-Pin	48-Pin
ACHIO0.0	4	—
ACHIO0.1	5	—
ACHIO0.2	3	—
ACHIO0.3	6	—
ACHIO0.4	07	43
ACHIO0.5	08	42
ACHIO0.6	7	40
ACHIO0.7	1	41
ALB	16	34
ANGND	66	44
BHE	37	13
CLKOUT	13	—
EA	7	39
EXTINT0.2	83	47
HSL0	84	3
HSL1	23	4
HSL0H0.4	32	2
HSL0H0.5	21	6
H0.0	20	3
H0.1	40	8
H0.2	44	9
H0.3	43	10
H0.4H0.2	22	5
H0.4H0.3	21	6
INST	12	—
NMI	7	—
PWM0.2	99	13
P0.0ACH0	4	—
P0.1ACH0	5	—
P0.2ACH0	3	—
P0.3ACH0	6	—
P0.4ACH0	07	43
P0.5ACH0	08	42
P0.6ACH0	7	40
P0.7ACH0	1	41
P1.0	20	—

Name	66-Pin	48-Pin
P1.1	28	—
P1.2	27	—
P1.3	26	—
P1.4	22	—
P1.5	48	—
P1.6	47	—
P1.7	46	—
P1.0TXD	60	—
P1.0RXD	61	—
P1.0EXTINT	83	47
P1.0TCLK	84	—
P1.0TSTR	36	—
P1.0PWM	39	13
P2.6	42	—
P2.7	40	—
P2.0ADO	18	33
P2.1ADI	19	31
P2.2ADO	20	30
P2.3ADO	21	29
P2.4ADI	22	28
P2.5ADO	23	27
P2.6ADO	24	26
P2.7ADO	25	25
P4.0ADO	26	24
P4.1ADI	27	23
P4.2ADO	28	22
P4.3ADI	29	21
P4.4ADO	30	20
P4.5ADI	31	19
P4.6ADO	32	18
P4.7ADI	33	17
RD	17	33
READY	35	35
RESET	62	48
R0DM0.1	87	1
TEST	14	—
T0D0.0	80	2
T0CLK0.2	34	—
T0STR0.4	36	—
VBR	41	12
VCC	9	—
VDD	64	38
VRP	65	42
VSS	10	11
VSS	42	37
WR	38	14
XTAL1	11	36
XTAL2	12	35

The following pins are not bonded out in the 48-pin package:  
P1.0 through P1.7, P0.0 through P0.7, P2.3, P2.4, P2.5, P2.6, P2.7, CLKOUT, INST, T0CLK0.2, T0STR0.4.

---

# MCS<sup>®</sup>-96 Software Design Information

---

**3**



Design Information  
MCS® 9d Software

3

### 3.0. INTRODUCTION

This section provides information which will primarily interest those who must write programs to execute in the 8096. Several other sources of information are currently available which will also be of interest:

#### **MCS®-96 MACRO ASSEMBLER USER'S GUIDE**

Order Number 122048-001

#### **MCS-96 UTILITIES USER'S GUIDE**

Order Number 122049-001

#### **MCS-96 MACRO ASSEMBLER AND UTILITIES POCKET REFERENCE**

Order Number 122050-001

Throughout this chapter short segments of code are used to illustrate the operation of the device. For these sections it has been assumed that a set of temporary registers have been predeclared. The names of these registers have been chosen as follows:

AX, BX, CX, and DX are 16 bit registers.

AL is the low byte of AX, AH is the high byte.

BL is the low byte of BX.

CL is the low byte of CX.

DL is the low byte of DX.

These are the same as the names for the general data registers used in 8086. It is important to note, however, that in the 8096, these are not dedicated registers but merely the symbolic names assigned by the programmer to an eight byte register within onboard register file.

### 3.1. OPERAND TYPES

The MCS®-96 architecture provides support for a variety of data types which are likely to be useful in a control application. In the discussion of these operand types that follows, the names adopted by the PLM-96 programming language will be used where appropriate. To avoid confusion the name of an operand type will be capitalized. A "BYTE" is an unsigned eight bit variable; a "byte" is an eight bit unit of data of any type.

#### **3.1.1. Bytes**

BYTES are unsigned 8-bit variables which can take on the values between 0 and 255. Arithmetic and relational operators can be applied to BYTE operands but the result must be interpreted in modulo 256 arithmetic. Logical operations on BYTES are applied bitwise. Bits within BYTES are labeled from 0 to 7 with 0 being the least significant bit. There are no alignment restrictions for BYTES so they may be placed anywhere in the MCS-96 address space.

#### **3.1.2. Words**

WORDS are unsigned 16-bit variables which can take on the values between 0 and 65535. Arithmetic and relational

operators can be applied to WORD operands but the result must be interpreted modulo 65536. Logical operations on WORDS are applied bitwise. Bits within words are labeled from 0 to 15 with 0 being the least significant bit. WORDS must be aligned at even byte boundaries in the MCS-96 address space. The least significant byte of the WORD is in the even byte address and the most significant byte is in the next higher (odd) address. The address of a word is the address of its least significant byte.

#### **3.1.3. Short-Integers**

SHORT-INTEGERS are 8-bit signed variables which can take on the values between -128 and +127. Arithmetic operations which generate results outside of the range of a SHORT-INTEGER will set the overflow indicators in the program status word. The actual numeric result returned will be the same as the equivalent operation on BYTE variables. There are no alignment restrictions on SHORT-INTEGERS so they may be placed anywhere in the MCS-96 address space.

#### **3.1.4. Integers**

INTEGERS are 16-bit signed variables which can take on the values between -32,768 and 32,767. Arithmetic operations which generate results outside of the range of an INTEGER will set the overflow indicators in the program status word. The actual numeric result returned will be the same as the equivalent operation on WORD variables. INTEGERS conform to the same alignment and addressing rules as do WORDS.

#### **3.1.5. Bits**

BITS are single-bit operands which can take on the Boolean values of true and false. In addition to the normal support for bits as components of BYTE and WORD operands, the 8096 provides for the direct testing of any bit in the internal register file. The MCS-96 architecture requires that bits be addressed as components of BYTES or WORDS, it does not support the direct addressing of bits that can occur in the MCS-51 architecture.

#### **3.1.6. Double-Words**

DOUBLE-WORDS are unsigned 32-bit variables which can take on the values between 0 and 4,294,967,295. The MCS-96 architecture provides direct support for this operand type only for shifts and as the dividend in a 32 by 16 divide and the product of a 16 by 16 multiply. For these operations a DOUBLE-WORD variable must reside in the on-board register file of the 8096 and be aligned at an address which is evenly divisible by 4. A DOUBLE-WORD operand is addressed by the address of its least significant byte. DOUBLE-WORD operations which are not directly supported can be easily implemented with two WORD operations. For consistency with INTEL provided software the user should adopt the conventions for addressing DOUBLE-WORD operands which are discussed in section 3.5.

### 3.1.7. Long-Integers

LONG-INTEGERS are 32-bit signed variables which can take on the values between -2,147,483,648 and 2,147,483,647. The MCS-96 architecture provides direct support for this data type only for shifts and as the dividend in a 32 by 16 divide and the product of a 16 by 16 multiply.

LONG-INTEGERS can also be normalized. For these operations a LONG-INTEGER variable must reside in the onboard register file of the 8096 and be aligned at an

address which is evenly divisible by 4. A LONG-INTEGER is addressed by the address of its least significant byte.

LONG-INTEGER operations which are not directly supported can be easily implemented with two INTEGER operations. For consistency with Intel provided software, the user should adopt the conventions for addressing LONG operands which are discussed in section 3.5.

## 3.2. OPERAND ADDRESSING

Operands are accessed within the address space of the 8096 with one of six basic addressing modes. Some of the details of how these addressing modes work are hidden by the assembly language. If the programmer is to take full advantage of the architecture, it is important that these details be understood. This section will describe the addressing modes as they are handled by the hardware. At the end of this section the addressing modes will be de-

scribed as they are seen through the assembly language. The six basic addressing modes which will be described are termed register-direct, indirect, indirect with auto-increment, immediate, short-indexed, and long-indexed. Several other useful addressing operations can be achieved by combining these basic addressing modes with specific registers such as the ZERO register or the stack pointer.

### 3.2.1. Register-direct References

The register-direct mode is used to directly access a register from the 256 byte on-board register file. The register is selected by an 8-bit field within the instruction and

register address must conform to the alignment rules for the operand type. Depending on the instruction, up to three registers can take part in the calculation.

#### Examples

ADD	AX,BX,CX	: AX = BX + CX
MUL	AX,BX	: AX = AX * BX
INCB	CL	: CL = CL + 1

### 3.2.2. Indirect References

The indirect mode is used to access an operand by placing its address in a WORD variable in the register file. The calculated address must conform to the alignment rules for the operand type. Note that the indirect address can refer to an operand anywhere within the address space of

the 8096, including the register file. The register which contains the indirect address is selected by an eight bit field within the instruction. An instruction can contain only one indirect reference and the remaining operands of the instruction (if any) must be register-direct references.

#### Examples

LD	AX,[AX]	: AX = MEM _ WORD(AX)
ADDB	AL,BL,[CX]	: AL = BL + MEM _ BYTE(CX)
POP	[AX]	: MEM _ WORD(AX) := MEM _ WORD(SP); SP = SP + 2

### 3.2.3. Indirect with Auto-increment References

This addressing mode is the same as the indirect mode except that the WORD variable which contains the indirect address is incremented *after* it is used to address the operand. If the instruction operates on BYTES or SHORT-

INTEGERS the indirect address variable will be incremented by one, if the instruction operates on WORDS or INTEGERS the indirect address variable will be incremented by two.

#### Examples

LD	AX,[BX] +	: AX = MEM _ WORD(BX); BX = BX + 2
ADD	AL,BL,[CX] +	: AL = AL + BL + MEM _ BYTE(CX); CX = CX + 1
PUSH	[AX] +	: SP = SP - 2; MEM _ WORD(SP) = MEM _ WORD(AX); AX = AX + 2

### 3.2.4. Immediate References

This addressing mode allows an operand to be taken directly from a field in the instruction. For operations on BYTE or SHORT-INTEGER operands this field is eight bits wide, for operations on WORD or INTEGER operands the field is 16 bits wide.

#### Examples

```
ADD AX,#340 ; AX:=AX+340
PUSH #1234H ; SP:=SP-2; MEM[SP]=1234H
DIVB AX,#10 ; AL:=AX/10; AH:=AX MOD 10
```

and the field is 16 bits wide. An instruction can contain only one immediate reference and the remaining operand(s) must be register-direct references.

### 3.2.5. Short-indexed References

In this addressing mode an eight bit field in the instruction selects a WORD variable in the register file which is assumed to contain an address. A second eight bit field in the instruction stream is sign-extended and summed with the WORD variable to form the address of the operand which will take part in the calculation. Since the

eight bit field is sign-extended the effective address can be up to 128 bytes before the address in the WORD variable and up to 127 bytes after it. An instruction can contain only one short-indexed reference and the remaining operand(s) must be register-direct references.

#### Examples

```
LD AX,12[BX] ; AX:=MEM[WORD(BX)+12]
MULB AX,BL,3[CX] ; AX:=BL*MEM[BYTE(CX)+3]
```

### 3.2.6. Long-indexed References

This addressing mode is like the short-indexed mode except that a 16-bit field is taken from the instruction and added to the WORD variable to form the address of the

operand. No sign extension is necessary. An instruction can contain only one long-indexed reference and the remaining operand(s) must be register-direct references.

#### Examples

```
AND AX,BX,TABLE[CX] ; AX:=BX AND MEM[WORD(TABLE)+CX]
ST AX,TABLE[BX] ; MEM[WORD(TABLE)+BX]:=AX
ADDB AL,BL,LOOKUP[CX] ; AL:=BL + MEM[BYTE(LOOKUP)+CX]
```

### 3.2.7. ZERO Register Addressing

The first two bytes in the register file are fixed at zero by the 8096 hardware. In addition to providing a fixed source of the constant zero for calculations and comparisons, this register can be used as the WORD variable in a long-

indexed reference. This combination of register selection and address mode allows any location in memory to be addressed directly.

#### Examples

```
ADD AX,1234[0] ; AX:=AX + MEM[WORD(1234)]
POP 5678[0] ; MEM[WORD(5678)]:=MEM[WORD(SP)]
; SP:=SP+2
```

### 3.2.8. Stack Pointer Register Addressing

The system stack pointer in the 8096 can be accessed as register 18H of the internal register file. In addition to providing for convenient manipulation of the stack pointer, this also facilitates the accessing of operands in the stack. The top of the stack, for example, can be accessed by

using the stack pointer as the WORD variable in an indirect reference. In a similar fashion, the stack pointer can be used in the short-indexed mode to access data within the stack.

#### Examples

```
PUSH [SP] ; DUPLICATE TOP OF STACK
LD AX,2[SP] ; AX:=NEXT TO TOP
```



### 3.2.9. Assembly Language Addressing Modes

The 8096 assembly language simplifies the choice of addressing modes to be used in several respects:

**Direct Addressing.** The assembly language will choose between register-direct addressing and long-indexed with the ZERO register depending on where the operand is in memory. The user can simply refer to an operand by its symbolic name; if the operand is in the register file, a register-direct reference will be used, if the operand is elsewhere in memory, a long-indexed reference will be generated.

### 3.3 PROGRAM STATUS WORD

The program status word (PSW) is a collection of Boolean flags which retain information concerning the state of the user's program. The format of the PSW is shown in figure 3-1. The information in the PSW can be broken down into

15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
Z	N	V	VT	C	—	I	ST	<Interrupt Mask Reg>							

Figure 3-1. PSW Register

#### 3.3.1. Interrupt Flags

The lower eight bits of the PSW are used to individually mask the various sources of interrupt to the 8096. A logical '1' in these bit positions enables the servicing of the corresponding interrupt. These mask bits can be accessed as an eight bit byte (INT\_MASK — address 8) in the on-board register file. Bit 9 in the PSW is the global interrupt enable. If this bit is cleared then all interrupts will be locked out except for the Non Maskable Interrupt (NMI). Note that the various interrupts are collected in the INT\_PENDING register even if they are locked out. Execution of the corresponding service routines will proceed according to their priority when they become enabled. Further information on the interrupt structure of the 8096 can be found in sections 2.5 and 3.6.

#### 3.3.2. Condition Flags

The remaining bits in the PSW are set as side effects of instruction execution and can be tested by the conditional jump instructions.

**Z.** The Z (Zero) flag is set to indicate that the operation generated a result equal to zero. For the add-with-carry (ADDC) and subtract-with-borrow (SUBC) operations the Z flag is cleared if the result is non-zero but is never set. These two instructions are normally used in conjunction with the ADD and SUB instructions to perform multiple precision arithmetic. The operation of the Z flag for these instructions leaves it indicating the proper result for the entire multiple precision calculation.

**N.** The N (Negative) flag is set to indicate that the op-

**Indexed Addressing.** The assembly language will choose between short and long indexing depending on the value of the index expression. If the value can be expressed in eight bits then short indexing will be used, if it cannot be expressed in eight bits then long indexing will be used.

The use of these features of the assembly language simplifies the programming task and should be used wherever possible.

two basic categories; interrupt control and condition flags. The PSW can be saved in the system stack with a single operation (PUSHF) and restored in a like manner (POPF).

eration generated a negative result. Note that the N flag will be set to the algebraically correct state even if the calculation overflows.

**V.** The V (oVerflow) flag is set to indicate that the operation generated a result which is outside the range that can be expressed in the destination data type.

**VT.** The VT (oVerflow Trap) flag is set whenever the V flag is set but can only be cleared by an instruction which explicitly operates on it such as the CLRVT or JVT instructions. The operation of the VT flag allows for the testing for a possible overflow condition at the end of a sequence of related arithmetic operations. This is normally more efficient than testing the V flag after each instruction.

**C.** The C (Carry) flag is set to indicate the state of the arithmetic carry from the most significant bit of the ALU for an arithmetic operation or the state of the last bit shifted out of the operand for a shift. Arithmetic Borrow after a subtract operation is the complement of the C flag (i.e. if the operation generated a borrow then C=0).

**ST.** The ST (STicky bit) set to indicate that during a right shift a 1 has been shifted first into the C flag and then been shifted out. The ST flag is undefined after a multiply operation. The ST flag can be used along with the C flag to control rounding after a right shift. Consider multiplying two eight bit quantities and then scaling the result down to 12 bits:

```
MULUB  AX,CL,DL ; AX:=CL*DL
SHR    AX,#4     ; Shift right 4 places
```

If the C flag is set after the shift it indicates that the bits shifted off the end of operand were greater-than or equal-to one half the least significant bit (LSB) of the result. If the C flag is clear after the shift it indicates that the bits shifted off the end of the operand were less than half the LSB of the result. Without the ST flag, the rounding decision must be made on the basis of this information alone. (Normally the result would be rounded up if the C flag is set.) The ST flag allows a finer resolution in the rounding decision:

C ST	Value of the bits shifted off
0 0	Value = 0
0 1	$0 < \text{Value} < \frac{1}{2} \text{ LSB}$
1 0	Value = $\frac{1}{2} \text{ LSB}$
1 1	Value $> \frac{1}{2} \text{ LSB}$

Figure 3-2. Rounding Alternatives

Imprecise rounding can be a major source of error in a numerical calculation; use of the ST flag improves the options available to the programmer.

### 3.4 INSTRUCTION SET

The MCS-96 instruction set contains a full set of arithmetic and logical operations for the 8-bit data types BYTE and SHORT INTEGER and for the 16-bit data types WORD and INTEGER. The DOUBLE-WORD and LONG data types (32 bits) are supported for the products of 16 by 16 multiplies and the dividends of 32 by 16 divides and for shift operations. The remaining operations on 32 bit variables can be implemented by combinations of 16 bit operations. As an example the sequence:

ADD ADDC	AX,CX BX,DX	
1	1	PC ← PC + 16-bit offset
1	1	PC ← PC + 16-bit offset
1	1	PC ← (A)
1	1	SP ← SP - 1 (8B) ← PC
1	1	PC ← PC + 16-bit offset
1	1	SP ← SP - 2 (2B) ← PC
1	1	PC ← PC + 16-bit offset
0	0	PC ← (SP) SP ← SP + 2
1	1	PC ← PC + 8-bit offset
1	1	Jump if C = 1
1	1	Jump if C = 0

performs a 32 bit addition, and the sequence

SUB  
SUBC AX,CX  
BX,DX

performs a 32 bit subtraction. Operations on REAL (i.e. floating point) variables are not supported directly by the hardware but are supported by the floating point library for the 8096 (FPAL-96) which implements a single precision subset of the proposed IEEE standard for floating point operations. The performance of this software is significantly improved by the 8096 NORML instruction which normalizes a 32-bit variable and by the existence of the ST flag in the PSW.

In addition to the operations on the various data types, the 8096 supports conversions between these types. LDBZE (load byte zero extended) converts a BYTE to a WORD and LDBSE (load byte sign extended) converts a SHORT-INTEGER into an INTEGER. WORDS can be converted to DOUBLE-WORDS by simply clearing the upper WORD of the DOUBLE-WORD (CLR) and INTEGERS can be converted to LONGS with the EXT (sign extend) instruction.

The MCS-96 instructions for addition, subtraction, and comparison do not distinguish between unsigned words and signed integers. Conditional jumps are provided to allow the user to treat the results of these operations as either signed or unsigned quantities. As an example, the CMPB (compare byte) instruction is used to compare both signed and unsigned eight bit quantities. A JH (jump if higher) could be used following the compare if unsigned operands were involved or a JGT (jump if greater-than) if signed operands were involved.

Tables 3-1 and 3-2 summarize the operation of each of the instructions and Tables 3-3 and 3-4 give the opcode, byte count, and timing information for each of the instructions.

# MCS<sup>®</sup>-96 SOFTWARE DESIGN INFORMATION

Table 3-1. Instruction Summary

Mnemonic	Operands	Operation (Note 1)	Flags						Notes
			Z	N	C	V	VT	ST	
ADD/ADDB	2	$D \leftarrow D + A$	✓	✓	✓	✓	↑	—	
ADD/ADDB	3	$D \leftarrow B + A$	✓	✓	✓	✓	↑	—	
ADDC/ADDCB	2	$D \leftarrow D + A + C$	↓	✓	✓	✓	↑	—	
SUB/SUBB	2	$D \leftarrow D - A$	✓	✓	✓	✓	↑	—	
SUB/SUBB	3	$D \leftarrow B - A$	✓	✓	✓	✓	↑	—	
SUBC/SUBCB	2	$D \leftarrow D - A + C - 1$	↓	✓	✓	✓	↑	—	
CMP/CMPB	2	$D - A$	✓	✓	✓	✓	↑	—	
MUL/MULU	2	$D, D + 2 \leftarrow D * A$	—	—	—	—	—	?	2
MUL/MULU	3	$D, D + 2 \leftarrow B * A$	—	—	—	—	—	?	2
MULB/MULUB	2	$D, D + 1 \leftarrow D * A$	—	—	—	—	—	?	3
MULB/MULUB	3	$D, D + 1 \leftarrow B * A$	—	—	—	—	—	?	3
DIV/DIVU	2	$D \leftarrow (D, D + 2)/A$ $D + 2$ remainder	—	—	—	✓	↑	—	2
DIVB/DIVUB	2	$D \leftarrow (D, D + 1)/A$ $D + 1$ remainder	—	—	—	✓	↑	—	3
AND/ANDB	2	$D \leftarrow D \text{ and } A$	✓	✓	0	0	—	—	
AND/ANDB	3	$D \leftarrow B \text{ and } A$	✓	✓	0	0	—	—	
OR/ORB	2	$D \leftarrow D \text{ or } A$	✓	✓	0	0	—	—	
XOR/XORB	2	$D \leftarrow D \text{ (excl. or) } A$	✓	✓	0	0	—	—	
LD/LDB	2	$D \leftarrow A$	—	—	—	—	—	—	
ST/STB	2	$A \leftarrow D$	—	—	—	—	—	—	
LDBSE	2	$D \leftarrow A; D + 1 \leftarrow \text{SIGN}(A)$	—	—	—	—	—	—	3,4
LDBZE	2	$D \leftarrow A; D + 1 \leftarrow 0$	—	—	—	—	—	—	3,4
PUSH	1	$SP \leftarrow SP - 2; (SP) \leftarrow A$	—	—	—	—	—	—	
POP	1	$A \leftarrow (SP); SP \leftarrow SP + 2$	—	—	—	—	—	—	
PUSHF	0	$SP \leftarrow SP - 2; (SP) \leftarrow \text{PSW};$ $\text{PSW} \leftarrow 0000\text{H}$ $I \leftarrow 0$	0	0	0	0	0	0	
POPF	0	$\text{PSW} \leftarrow (SP); SP \leftarrow SP + 2$ $I \leftarrow \checkmark$	✓	✓	✓	✓	✓	✓	
SJMP	1	$PC \leftarrow PC + 11\text{-bit offset}$	—	—	—	—	—	—	5
LJMP	1	$PC \leftarrow PC + 16\text{-bit offset}$	—	—	—	—	—	—	5
BR(indirect)	1	$PC \leftarrow (A)$	—	—	—	—	—	—	
SCALL	1	$SP \leftarrow SP - 2; (SP) \leftarrow PC;$ $PC \leftarrow PC + 11\text{-bit offset}$	—	—	—	—	—	—	5
LCALL	1	$SP \leftarrow SP - 2; (SP) \leftarrow PC;$ $PC \leftarrow PC + 16\text{-bit offset}$	—	—	—	—	—	—	5
RET	0	$PC \leftarrow (SP); SP \leftarrow SP + 2$	—	—	—	—	—	—	
J(conditional)	1	$PC \leftarrow PC + 8\text{-bit offset}$	—	—	—	—	—	—	5
JC	1	Jump if $C = 1$	—	—	—	—	—	—	5
JNC	1	Jump if $C = 0$	—	—	—	—	—	—	5

## Note

1. If the mnemonic ends in "B", a byte operation is performed, otherwise a word operation is done. Operands D, B, and A must conform to the alignment rules for the required operand type. D and B are locations in the register file; A can be located anywhere in memory.
2. D, D + 2 are consecutive WORDS in memory; D is DOUBLE-WORD aligned.
3. D, D + 1 are consecutive BYTES in memory; D is WORD aligned.
4. Changes a byte to a word.
5. Offset is a 2's complement number.

Table 3-2. Instruction Summary

Mnemonic	Oper- ands	Operation (Note 1)	Flags							Notes
			Z	N	C	V	VT	ST		
JE	1	Jump if Z = 1	—	—	—	—	—	—	5	
JNE	1	Jump if Z = 0	—	—	—	—	—	—	5	
JGE	1	Jump if N = 0	—	—	—	—	—	—	5	
JLT	1	Jump if N = 1	—	—	—	—	—	—	5	
JGT	1	Jump if N = 0 and Z = 0	—	—	—	—	—	—	5	
JLE	1	Jump if N = 1 or Z = 1	—	—	—	—	—	—	5	
JH	1	Jump if C = 1 and Z = 0	—	—	—	—	—	—	5	
JNH	1	Jump if C = 0 or Z = 1	—	—	—	—	—	—	5	
JV	1	Jump if V = 1	—	—	—	—	—	—	5	
JNV	1	Jump if V = 0	—	—	—	—	—	—	5	
JVT	1	Jump if VT = 1; Clear VT	—	—	—	—	0	—	5	
JNVT	1	Jump if VT = 0; Clear VT	—	—	—	—	0	—	5	
JST	1	Jump if ST = 1	—	—	—	—	—	—	5	
JNST	1	Jump if ST = 0	—	—	—	—	—	—	5	
JBS	3	Jump if Specified Bit = 1	—	—	—	—	—	—	5,6	
JBC	3	Jump if Specified Bit = 0	—	—	—	—	—	—	5,6	
DJNZ	1	D ← D − 1; if D ≠ 0 then PC ← PC + 8-bit offset	—	—	—	—	—	—	5	
DEC/DECB	1	D ← D − 1	✓	✓	✓	✓	↑	—		
NEG/NEGB	1	D ← 0 − D	✓	✓	✓	✓	↑	—		
INC/INCB	1	D ← D + 1	✓	✓	✓	✓	↑	—		
EXT	1	D ← D; D + 2 ← Sign (D)	✓	✓	0	0	—	—	2	
EXTB	1	D ← D; D + 1 ← Sign (D)	✓	✓	0	0	—	—	3	
NOT/NOTB	1	D ← Logical Not (D)	✓	✓	0	0	—	—		
CLR/CLRB	1	D ← 0	1	0	0	0	—	—		
SHL/SHLB/SHLL	2	C ← msb — — — — lsb ← 0	✓	?	✓	✓	↑	—	7	
SHR/SHRB/SHRL	2	0 → msb — — — — lsb → C	✓	0	✓	0	—	✓	7	
SHRA/SHRAB/SHRAL	2	msb → msb — — — — lsb → C	✓	✓	✓	0	—	✓	7	
SETC	0	C ← 1	—	—	1	—	—	—		
CLRC	0	C ← 0	—	—	0	—	—	—		
CLRVT	0	VT ← 0	—	—	—	—	0	—		
RST	0	PC ← 2080H	0	0	0	0	0	0	8	
DI	0	Disable All Interrupts (I ← 0)	—	—	—	—	—	—		
EI	0	Enable All Interrupts (I ← 1)	—	—	—	—	—	—		
NOP	0	PC ← PC + 1	—	—	—	—	—	—		
SKIP	0	PC ← PC + 2	—	—	—	—	—	—		
NORML	2	Normalize (See sec 3.13.66)	✓	1	0	—	—	—	7	
TRAP	0	SP ← SP − 2; (SP) ← PC; PC ← (2010H)	—	—	—	—	—	—	9	

**Note**

1. If the mnemonic ends in "B", a byte operation is performed, otherwise a word operation is done. Operands D, B and A must conform to the alignment rules for the required operand type. D and B are locations in the register file; A can be located anywhere in memory.
5. Offset is a 2's complement number.
6. Specified bit is one of the 2048 bits in the register file.
7. The "L" (Long) suffix indicates double-word operation.
8. Initiates a Reset by pulling RESET low. Software should re-initialize all the necessary registers with code starting at 2080H.
9. The assembler will not accept this mnemonic.



# MCS®-96 SOFTWARE DESIGN INFORMATION

Table 3-3. Opcode and State Time Listing

MNEMONIC	OPERANDS	DIRECT			IMMEDIATE			INDIRECT®					INDEXED®					
		OPCODE	BYTES	STATE TIMES	OPCODE	BYTES	STATE TIMES	NORMAL			AUTO-INC.		SHORT			LONG		
								OPCODE	BYTES	STATE <sup>①</sup> TIMES	BYTES	STATE <sup>①</sup> TIMES	OPCODE	BYTES	STATE <sup>①</sup> TIMES	BYTES	STATE <sup>①</sup> TIMES	
ARITHMETIC INSTRUCTIONS																		
ADD	2	64	3	4	65	4	5	66	3	6/11	3	7/12	67	4	6/11	5	7/12	
ADD	3	44	4	5	45	5	6	46	4	7/12	4	8/13	47	5	7/12	6	8/13	
ADDB	2	74	3	4	75	3	4	76	3	6/11	3	7/12	77	4	6/11	5	7/12	
ADDB	3	54	4	5	55	4	5	56	4	7/12	4	8/13	57	5	7/12	6	8/13	
ADDC	2	A4	3	4	A5	4	5	A6	3	6/11	3	7/12	A7	4	6/11	5	7/12	
ADDCB	2	B4	3	4	B5	3	4	B6	3	6/11	3	7/12	B7	4	6/11	5	7/12	
SUB	2	68	3	4	69	4	5	6A	3	6/11	3	7/12	6B	4	6/11	5	7/12	
SUB	3	48	4	5	49	5	6	4A	4	7/12	4	8/13	4B	5	7/12	6	8/13	
SUBB	2	78	3	4	79	3	4	7A	3	6/11	3	7/12	7B	4	6/11	5	7/12	
SUBB	3	58	4	5	59	4	5	5A	4	7/12	4	8/13	5B	5	7/12	6	8/13	
SUBC	2	A8	3	4	A9	4	5	AA	3	6/11	3	7/12	AB	4	6/11	5	7/12	
SUBCB	2	B8	3	4	B9	3	4	BA	3	6/11	3	7/12	BB	4	6/11	5	7/12	
CMP	2	88	3	4	89	4	5	8A	3	6/11	3	7/12	8B	4	6/11	5	7/12	
CMPB	2	98	3	4	99	3	4	9A	3	6/11	3	7/12	9B	4	6/11	5	7/12	
MULU	2	6C	3	25	6D	4	26	6E	3	27/32	3	28/33	6F	4	27/32	5	28/33	
MULU	3	4C	4	26	4D	5	27	4E	4	28/33	4	29/34	4F	5	28/33	6	29/34	
MULUB	2	7C	3	17	7D	3	17	7E	3	19/24	3	20/25	7F	4	19/24	5	20/25	
MULUB	3	5C	4	18	5D	4	18	5E	4	20/25	4	21/26	5F	5	20/25	6	21/26	
MUL	2	②	4	29	②	5	30	②	4	31/36	4	32/37	②	5	31/36	6	32/37	
MUL	3	②	5	30	②	6	31	②	5	32/37	5	33/38	②	6	32/37	7	33/38	
MULB	2	②	4	21	②	4	21	②	4	23/28	4	24/29	②	5	23/28	6	24/29	
MULB	3	②	5	22	②	5	22	②	5	24/29	5	25/30	②	6	24/29	7	25/30	
DIVU	2	8C	3	25	8D	4	26	8E	3	28/32	3	29/33	8F	4	28/32	5	29/33	
DIVUB	2	9C	3	17	9D	3	17	9E	3	20/24	3	21/25	9F	4	20/24	5	21/25	
DIV	2	②	4	29	②	5	30	②	4	32/36	4	33/37	②	5	32/36	6	33/37	
DIVB	2	②	4	21	②	4	21	②	4	24/28	4	25/29	②	5	24/28	6	25/29	

## Notes:

- Long indexed and Indirect + instructions have identical opcodes with Short indexed and Indirect modes, respectively. The second byte of instructions using any indirect or indexed addressing mode specifies the exact mode used. If the second byte is even, use Indirect or Short Indexed. If it is odd, use Indirect + or Long indexed. In all cases the second byte of the instruction always specifies an even (word) location for the address referenced.
- Number of state times shown for internal/external operands.
- The opcodes for signed multiply and divide are the opcodes for the unsigned functions with an "FE" appended as a prefix.

MNEMONIC	OPERANDS	DIRECT			IMMEDIATE			INDIRECT <sup>②</sup>					INDEXED <sup>②</sup>				
		OPCODE	BYTES	STATE TIMES	OPCODE	BYTES	STATE TIMES	NORMAL			AUTO-INC.		SHORT			LONG	
								OPCODE	BYTES	STATE <sup>①</sup> TIMES	BYTES	STATE <sup>①</sup> TIMES	OPCODE	BYTES	STATE <sup>①</sup> TIMES	BYTES	STATE <sup>①</sup> TIMES
LOGICAL INSTRUCTIONS																	
AND	2	60	3	4	61	4	5	62	3	6/11	3	7/12	63	4	6/11	5	7/12
AND	3	40	4	5	41	5	6	42	4	7/12	4	8/13	43	5	7/12	6	8/13
ANDB	2	70	3	4	71	3	4	72	3	6/11	3	7/12	73	4	6/11	5	7/12
ANDB	3	50	4	5	51	4	5	52	4	7/12	4	8/13	53	5	7/12	6	8/13
OR	2	80	3	4	81	4	5	82	3	6/11	3	7/12	83	4	6/11	5	7/12
ORB	2	90	3	4	91	3	4	92	3	6/11	3	7/12	93	4	6/11	5	7/12
XOR	2	84	3	4	85	4	5	86	3	6/11	3	7/12	87	4	6/11	5	7/12
XORB	2	94	3	4	95	3	4	96	3	6/11	3	7/12	97	4	6/11	5	7/12
DATA TRANSFER INSTRUCTIONS																	
LD	2	A0	3	4	A1	4	5	A2	3	6/11	3	7/12	A3	4	6/11	5	7/12
LDB	2	B0	3	4	B1	3	4	B2	3	6/11	3	7/12	B3	4	6/11	5	7/12
ST	2	C0	3	4	—	—	—	C2	3	7/11	3	8/12	C3	4	7/11	5	8/12
STB	2	C4	3	4	—	—	—	C6	3	7/11	3	8/12	C7	4	7/11	5	8/12
LDBSE	2	BC	3	4	BD	3	4	BE	3	6/11	3	7/12	BF	4	6/11	5	7/12
LDBZE	2	AC	3	4	AD	3	4	AE	3	6/11	3	7/12	AF	4	6/11	5	7/12
STACK OPERATIONS (internal stack)																	
PUSH	1	C8	2	8	C9	3	8	CA	2	11/15	2	12/16	CB	3	11/15	4	12/16
POP	1	CC	2	12	—	—	—	CE	2	14/18	2	14/18	CF	3	14/18	4	14/18
PUSHF	0	F2	1	8													
POPF	0	F3	1	9													
STACK OPERATIONS (external stack)																	
PUSH	1	C8	2	12	C9	3	12	CA	2	15/19	2	16/20	CB	3	15/19	4	16/20
POP	1	CC	2	14	—	—	—	CE	2	16/20	2	16/20	CF	3	16/20	4	16/20
PUSHF	0	F2	1	12													
POPF	0	F3	1	13													

JUMPS AND CALLS							
MNEMONIC	OPCODE	BYTES	STATES	MNEMONIC	OPCODE	BYTES	STATES
LJMP	E7	3	8	LCALL	EF	3	13/16 <sup>⑤</sup>
SJMP	20-27 <sup>④</sup>	2	8	SCALL	28-2F <sup>④</sup>	2	13/16 <sup>⑤</sup>
BR[ ]	E3	2	8	RET	F0	1	12/16 <sup>⑤</sup>
				TRAP <sup>③</sup>	F7	1	

**Notes:**

- ① Number of state times shown for internal/external operands.
- ② The assembler does not accept this mnemonic.
- ③ The least significant 3 bits of the opcode are concatenated with the following 8 bits to form an 11-bit, 2's complement, offset for the relative call or jump.
- ④ State times for stack located internal/external.
- ⑤ The assembler uses the generic jump mnemonic (BR) to generate this instruction.

# MCS®-96 SOFTWARE DESIGN INFORMATION

Table 3-4. CONDITIONAL JUMPS

All conditional jumps are 2 byte instructions. They require 8 state times if the jump is taken, 4 if it is not.							
MNEMONIC	OPCODE	MNEMONIC	OPCODE	MNEMONIC	OPCODE	MNEMONIC	OPCODE
JC	DB	JE	DF	JGE	D6	JGT	D2
JNC	D3	JNE	D7	JLT	DE	JLE	DA
JH	D9	JV	DD	JVT	DC	JST	D8
JNH	D1	JNV	D5	JNVT	D4	JNST	D0

## JUMP ON BIT CLEAR OR BIT SET

These instructions are 3-byte instructions. They require 9 state times if the jump is taken, 5 if it is not.								
BIT NUMBER								
MNEMONIC	0	1	2	3	4	5	6	7
JBC	30	31	32	33	34	35	36	37
JBS	38	39	3A	3B	3C	3D	3E	3F

## LOOP CONTROL

DJNZ	OPCODE EO;	3 BYTES;	5/9 STATE TIMES (NOT TAKEN/TAKEN)
------	------------	----------	-----------------------------------

## SINGLE REGISTER INSTRUCTIONS

MNEMONIC	OPCODE	BYTES	STATES	MNEMONIC	OPCODE	BYTES	STATES
DEC	05	2	4	EXT	06	2	4
DECB	15	2	4	EXTB	16	2	4
NEG	03	2	4	NOT	02	2	4
NEGB	13	2	4	NOTB	12	2	4
INC	07	2	4	CLR	01	2	4
INCB	17	2	4	CLRB	11	2	4

## SHIFT INSTRUCTIONS

INSTR MNEMONIC	WORD		INSTR MNEMONIC	BYTE		INSTR MNEMONIC	DBL WD		STATE TIMES
	OP	B		OP	B		OP	B	
SHL	09	3	SHLB	19	3	SHLL	0D	3	7 + 1 PER SHIFT⑦
SHR	08	3	SHRB	18	3	SHRL	0C	3	7 + 1 PER SHIFT⑦
SHRA	0A	3	SHRAB	1A	3	SHRAL	0E	3	7 + 1 PER SHIFT⑦

## SPECIAL CONTROL INSTRUCTIONS

MNEMONIC	OPCODE	BYTES	STATES	MNEMONIC	OPCODE	BYTES	STATES
SETC	F9	1	4	DI	FA	1	4
CLRC	F8	1	4	EI	FB	1	4
CLRVT	FC	1	4	NOP	FD	1	4
RST	FF	1	16	SKIP	00	2	4

## NORMALIZE

NORML	0F	3	11 + 1 PER SHIFT
-------	----	---	------------------

### Notes:

⑥ This instruction takes 2 states to pull RST low, then holds it low for 2 states to initiate a reset. The reset takes 12 states, at which time the program restarts at location 2080H.

⑦ Execution will take at least 8 states, even for 0 shift.

### 3.5. SOFTWARE STANDARDS AND CONVENTIONS

For a software project of any size it is a good idea to modularize the program and to establish standards which control the communication between these modules. The nature of these standards will vary with the needs of the final application. A common component of all of these standards, however, must be the mechanism for passing parameters to procedures and returning results from procedures. In the absence of some overriding consideration which prevents their use, it is suggested that the user conform to the conventions adopted by the PLM-96 programming language for procedure linkage. It is a very usable standard for both the assembly language and PLM-96 environment and it offers compatibility between these environments. Another advantage is that it allows the user access to the same floating point arithmetics library that PLM-96 uses to operate on REAL variables.

#### 3.5.1. Register Utilization

The MCS-96 architecture provides a 256 byte register file. Some of these registers are used to control register-mapped I/O devices and for other special functions such as the ZERO register and the stack pointer. The remaining bytes in the register file, some 230 of them, are available for allocation by the programmer. If these registers are to be used effectively some overall strategy for their allocation must be adopted. PLM-96 adopts the simple and effective strategy of allocating the eight bytes between addresses 1CH and 23H as temporary storage. The starting address of this region is called PLMREG. The remaining area in the register file is treated as a segment of memory which is allocated as required.

#### 3.5.2. Addressing 32-bit Operands

These operands are formed from two adjacent 16-bit words in memory. The least significant word of the double word is always in lower address, even when the data is in the stack (which means that the most significant word must be pushed into the stack first). A double word is addressed by the address of its least significant byte. Note that the hardware supports some operations on double words (e.g. normalize and divide). For these operations the double word must be in the internal register file and must have an address which is evenly divisible by four.

#### 3.5.3. Subroutine Linkage

Parameters are passed to subroutines in the stack. Parameters are pushed into the stack in the order that they are encountered in the scanning of the source text. Eight-bit parameters (BYTES or SHORT-INTEGERS) are pushed into the stack with the high order byte undefined. Thirty-two bit parameters (LONG-INTEGERS, DOUBLE-WORDS, and REALS) are pushed into the stack as two 16 bit values; the most significant half of the parameter is pushed into the stack first.

As an example, consider the following PLM-96 procedure:

```
example__procedure: PROCEDURE (param1,param2,param3);
  DECLARE param1 BYTE,
  param2 DWORD,
  param3 WORD;
```

When this procedure is entered at run time the stack will contain the parameters in the following order:

????? :	param1	
high word of	param2	
low word of	param2	
	param3	
	return address	← Stack_pointer

Figure 3-3. Stack Image

If a procedure returns a value to the calling code (as opposed to modifying more global variables) then the result is returned in the variable PLMREG. PLMREG is viewed as either an 8, 16 or 32 bit variable depending on the type of the procedure.

The standard calling convention adopted by PLM-96 has several key features:

- Procedures can always assume that the eight bytes of register file memory starting at PLMREG can be used as temporaries within the body of the procedure.
- Code which calls a procedure must assume that the eight bytes of register file memory starting at PLMREG are modified by the procedure.
- The Program Status Word (PSW-see section 3.3) is not saved and restored by procedures so the calling code must assumed that the condition flags (Z,N,V,VT,C, and ST) are modified by the procedure.
- Function results from procedures are always returned in the variable PLMREG.

PLM-96 allows the definition of INTERRUPT procedures which are executed when a predefined interrupt occurs. These procedures do not conform to the rules of a normal procedure. Parameters cannot be passed to these procedures and they cannot return results. Since they can execute essentially at any time (hence the term interrupt), these procedures must save the PSW and PLMREG when they are entered and restore these values before they exit.

### 3.6. USING THE INTERRUPT SYSTEM

Processing interrupts is an integral part of almost any control application. The 8096 allows the program to manage interrupt servicing in an efficient and flexible manner. Software running in the 8096 exerts control over the interrupt hardware at several levels.



### 3.6.1. Global Lockout

The processing of interrupts can be enabled or disabled by setting or clearing the I bit in the PSW. This is accomplished by the EI (Enable Interrupts) and DI (Disable Interrupts) instructions. Note that the I bit only controls the actual servicing of interrupts; interrupts that occur during periods of lockout will be held in the pending register and serviced on a prioritized basis when the lockout period ends.

### 3.6.2. Pending Interrupt Register

When the hardware detects one of the eight interrupts it sets the corresponding bit in the pending interrupt register (INT\_PENDING-register 09H). This register, which has the same bit layout as the interrupt mask register (see next section), can be read or modified as a byte register. This register can be read to determine which of the interrupts are pending at any given time or modified to either clear pending interrupts or generate interrupts under software control. Any software which modifies the INT\_PENDING register should ensure that the entire operation is indivisible. The easiest way of doing this is to use the logical instructions in the two or three operand format, as examples:

```
ANDB INT_PENDING,#11111101B
ORB INT_PENDING,#00000010B
; Clears the A/D interrupt
; Sets the A/D interrupt
```

If the required modification to INT\_PENDING cannot be accomplished with one instruction then a critical region should be established and the INT\_PENDING register modified from within this region (see section 3.6.5).

### 3.6.3. Interrupt Mask Register

Individual interrupts can be enabled or disabled by setting or clearing bits in the interrupt mask register (INT\_MASK-register 08H). The format of this register is shown in figure 3-4.

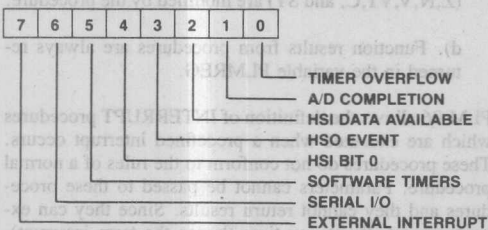


Figure 3-4. Interrupt Mask Register

The INT\_MASK register can be read or written as a byte register. A one in any bit position will enable the corresponding interrupt source and a zero will disable the source. The individual masks act like the global lockout in that they only control the servicing of the interrupt; the hardware will save any interrupts that occur in the pending register even if the interrupt mask bit is cleared. The INT\_MASK register also can be accessed as the lower

eight bits of the PSW so the PUSHF and POPF instructions save and restore the INT\_MASK register as well as the global interrupt lockout and the arithmetic flags.

### 3.6.4. Interrupt Vectors

The 8096 has eight sources of hardware interrupt, each with its own priority and interrupt vector location. Table 3-5 shows the interrupt sources, their priority, and their vector locations. See section 2.5 for a discussion of the various interrupt sources.

Table 3-5. Interrupt Vector Information

Source	Priority	Vector
Timer Overflow	0-Lowest	2000H
A/D Completion	1	2002H
HSI Data Available	2	2004H
HSO Execution	3	2006H
HSI.O	4	2008H
Software timers	5	200AH
Serial I/O	6	200CH
External Interrupt	7-Highest	200EH

The programmer must initialize the interrupt vector table with the starting addresses of the appropriate interrupt service routine. It would be a good idea to vector any interrupts that are not used in the system to an error handling routine.

The priorities given in the table give the hardware enforced priorities for these interrupts. This priority controls the order in which pending interrupts are passed to the software via interrupt-calls. The software can implement its own priority structure by controlling the mask register (INT\_MASK-register 08H). To see how this is done consider the case of a serial I/O service routine which must run at a priority level which is lower than the HSI data available interrupt but higher than any other source. The "preamble" and exit code for this interrupt service routine would look like this:

```
serial_io_isr:
    PUSHF                ; Save the PSW
                        ; (Includes INT_MASK)
    LDB INT_MASK,#00000100B
    EI                   ; Enable interrupts again
    ;
    ;
    ; } Service the interrupt
    ;
    POPF                ; Restore the PSW
    RET
```

Note that location 200CH in the interrupt vector table would have to be loaded with the value of the label serial\_io\_isr and the interrupt be enabled for this routine to execute.

There is an interesting chain of instruction side-effects which makes this (or any other) 8096 interrupt service routine execute properly:

a). After the hardware decides to process an interrupt it generates and executes a special interrupt-call instruction which pushes the current program counter onto the stack and then loads the program counter with the contents of the vector table entry corresponding to the interrupt. The hardware will not allow another interrupt to be serviced immediately following the interrupt-call. This guarantees that once the interrupt-call starts the first instruction of the interrupt service routine will execute.

b). The PUSHF instruction, which is now guaranteed to execute, saves the PSW in the stack and then clears the PSW. The PSW contains, in addition to the arithmetic flags, the INT\_MASK register and the global enable flag (I). The hardware will not allow an interrupt following a PUSHF instruction and by the time the LD instruction starts all of the interrupt enable flags will be cleared. Now there is guaranteed execution of the LD INT\_MASK instruction.

c). The LD INT\_MASK instruction enables those interrupts that the programmer chooses to allow to interrupt the serial I/O interrupt service routine. In this example only the HSI data available interrupt will be allowed to do this but any interrupt or combination of interrupts could be enabled at this point, even the serial interrupt. It is the loading of the INT\_MASK register which allows the software to establish its own priorities for interrupt servicing independently from those that the hardware enforces.

d). The EI instruction reenables the processing of interrupts.

e). The actual interrupt service routine executes within the priority structure established by the software.

f). At the end of the service routine the POPF instruction restores the PSW to its state when the interrupt-call occurred. The hardware will not allow interrupts to be processed following a POPF instruction so the execution of the last instruction (RET) is guaranteed before further interrupts can occur. The reason that this RET instruction must be protected in this fashion is that it is quite likely that the POPF instruction will reenables an interrupt which is already pending. If this interrupt were serviced before the RET instruction, then the return address to the code that was executing when the original interrupt occurred would be left on the stack. While this does not present a problem to the program flow, it could result in a stack overflow if interrupts are occurring at a high frequency. The POPF instruction also pops the INT\_MASK register (part of the PSW), so any changes made to this register during a routine which ends with a POPF will be lost.

Notice that the "preamble" and exit code for the interrupt service routine does not include any code for saving or restoring registers. This is because it has been assumed that the interrupt service routine has been allocated its own private set of registers from the on-board register file. The availability of some 230 bytes of register storage makes this quite practical.

### 3.6.5. Critical Regions

Interrupt service routines must share some data with other routines. Whenever the programmer is coding those sections of code which access these shared pieces of data, great care must be taken to ensure that the integrity of the data is maintained. Consider clearing a bit in the interrupt pending register as part of a non-interrupt routine:

```
LDB     AL,INT_PENDING
ANDB    AL,#bit_mask
STB     AL,INT_PENDING
```

This code works if no other routines are operating concurrently, but will cause occasional but serious problems if used in a concurrent environment. (All programs which make use of interrupts must be considered to be part of a concurrent environment.) To demonstrate this problem, assume that the INT\_PENDING register contains 00001111B and bit 3 (HSO event interrupt pending) is to be reset. The code does work for this data pattern but what happens if an HSI interrupt occurs somewhere between the LDB and the STB instructions? Before the LDB instruction INT\_PENDING contains 00001111B and after the LDB instruction so does AL. If the HSI interrupt service routine executes at this point then INT\_PENDING will change to 00001011B. The ANDB changes AL to 00000111B and the STB changes INT\_PENDING to 00000111B. It should be 00000011B. This code sequence has managed to generate a false HSI interrupt! The same basic process can generate an amazing assortment of problems and headaches. These problems can be avoided by assuring mutual exclusion which basically means that if more than one routine can change a variable, then the programmer must ensure exclusive access to the variable during the entire operation on the variable.

In many cases the instruction set of the 8096 allows the variable to be modified with a single instruction. The code in the above example can be implemented with a single instruction:

```
ANDB    INT_PENDING,#bit_mask
```

Instructions are indivisible so mutual exclusion is ensured in this case. For more complex situations, such a simple solution is not available and the programmer must create what is termed a critical region in which it is safe to modify the variable. One way to do this is to simply disable interrupts with a DI instruction, perform the modification, and then re-enable interrupts with an EI instruction. The problem with this approach is that it leaves the interrupts enabled even if they were not enabled at the start. A better solution is to enter the critical region with a PUSHF instruction which saves the PSW and also clears

the interrupt enable flags. The region can then be terminated with a POPF instruction which returns the interrupt enable to the state it was in before the code sequence. It should be noted that some system configurations might require more protection to form a critical region. An example is a system in which more than one processor has access to a common resource such as memory or external I/O devices.

### 3.7. I/O PROGRAMMING CONSIDERATIONS

The on-board I/O devices are, for the most part, simple to program. There are some areas of potential confusion which need to be addressed:

#### 3.7.1. Programming the I/O Ports

Some of the on-board I/O ports can be used as both input and output pins (e.g. Port 1). When the processor writes to the pins of these ports it actually writes into a register which in turn drives the port pin. When the processor reads these ports, it senses the status of the pin directly. If a port pin is to be used as an input then the software should write a one to that pin, this will cause the low-impedance pull-down device to turn off and leave the pin pulled up with a relatively high impedance pull-up device which can be easily driven down by the device driving the input. If some pins of a port are to be used as inputs and some are to be used as outputs the programmer should be careful when writing to the port. Consider using P1.0 as an input and then trying to toggle P1.1 as an output:

```
ORB  IOPORT1,#00000001B ; Set P1.0 for input
XORB IOPORT1,#00000010B ; Complement P1.1
```

The first instruction will work as expected but two problems can occur when the second instruction executes. The first is that even though P1.1 is being driven high by the 8096 it is possible that it is being held low externally. This typically happens when the port pin is used to drive the base of an NPN transistor which in turn drives whatever there is in the outside world which needs to be toggled. The base of the transistor will clamp the port pin to the transistor's Vbe above ground, typically 0.7 volts. The 8096 will input this value as a zero even if a one has been written to the port pin. When this happens the XORB instruction will always write a one to the port pin and it will not toggle. The second problem, which is related to the first one, is that if P1.0 happens to be driven to a zero when Port 1 is read by the XORB instruction then the XORB will write a zero to P1.0 and it will no longer be useable as an input. The first problem can best be solved by the external driver design. A series resistor between the port pin and the base of the transistor often works. The second problem can be solved in the software fairly easily:

```
LDB  AL,IOPORT1
XORB AL,#010B
ORB  AL,#001B
STB  AL,IOPORT1
```

A software solution to both problems is to keep a byte in RAM as an image of the data to be output to the port; any time the software wants to modify the data on the port it can then modify the image byte and then copy it to the port.

#### 3.7.2. Reading the I/O Status Register 1

This status register contains a collection of status flags which relate to the timer and high speed I/O functions (see section 2.12.5). It can be accessed as register 16H in the on-board register file. The layout of this register is shown in figure 3-5.

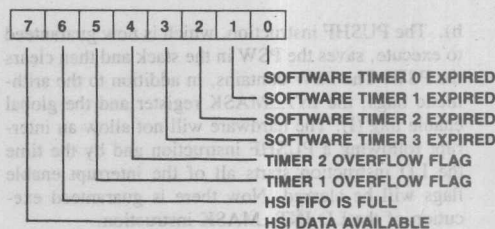


Figure 3-5. I/O Status Register 1

Whenever the processor reads this register all of the time-related flags (bits 5 through 0) are cleared. This applies not only to explicit reads such as:

```
LDB  AL,IOS1
```

but also to implicit reads such as:

```
JB  IOS1.3,somewhere — else
```

which jumps to somewhere — else if bit 3 of IOS1 is set. In most cases this situation can best be handled by having a byte in the register file which is used to maintain an image of lower five bits of the register. Any time a hardware timer interrupt or a HSO software timer interrupt occurs the byte can be updated:

```
ORB  IOS1 — image,IOS1
```

leaving IOS1 — image containing all the flags that were set before plus all the new flags that were read and cleared from IOS1. Any other routine which needs to sample the flags can safely check IOS1 — image. Note that if these routines need to clear the flags that they have acted on then the modification of IOS1 — image must be done from inside a critical region (see section 3.6.5).

#### 3.7.3. Sending Commands to the HSO Unit

Commands are sent to the HSO unit via a byte and then a word write operation:

```
LDB  HSO — COMMAND,#what — to — do
ADD  HSO — TIME,TIMER1,#when — to — do — it
```



The command is actually accepted when the HSO — TIME register is written. It is important to ensure that this code piece is not interrupted by any interrupt service routine which might also send a command to the HSO unit. If this happens the HSO will know when to do it but not know what to do when it's time to do it. In many systems this becomes a null problem because HSO commands are only issued from one place in the code. If this is not the case then a critical region must be established and the two instructions executed from within this region (see section 3.6.5).

Commands in the holding register will not execute even if their time tag is reached. Commands must be in the CAM for this to occur. Flags are available in IOS0 which indicate the holding register is empty (IOS0.7) or that both the holding register is empty *and* the CAM is not full (IOS0.6). The programmer should carefully decide which of these two flags is the best to use for each application.

It is possible to enter commands into the CAM which never execute. This occurs if TIMER2 has been set up as a variable modulo counter and a command is entered with a time tag referenced to TIMER2 which has a value that TIMER2 never reaches. The inaccessible command will never execute and continue to take up room in the CAM until either the system is reset or the program allows TIMER2 increment up to the value stored in the time tag. Note that commands cannot be flushed from the CAM without being executed but that they can be cancelled. This is accomplished by setting the opposite command in the CAM to execute at the same time tag as the command to be cancelled. Since internal events are not synchronized to Timer 1, it is not possible to cancel them. If, as an example, a command has been issued to set HSO.1 when TIMER1 = 1234 then entering a second command which clears HSO.1 when TIMER1 = 1234 will result in a no-operation on HSO.1. Both commands will remain in the CAM until TIMER1 = 1234.

### 3.7.4. High Speed I/O Interrupts

The HSO unit can generate two types of interrupts. The HSO execution interrupt (vector = (2006H)) is generated (if enabled) for HSO commands which operate on one of the six HSO pins. The other HSO interrupt is the Software Timer interrupt (vector = (200AH)) which is generated (if enabled) for any other HSO command (e.g. triggering the A/D, resetting Timer2 or generating a software time delay).

There are also two interrupts associated with the HSI unit. The HSI data available interrupt (vector = (2004H)) is generated if there is data in the HSI FIFO that the program should read. The other HSI related interrupt is the HSI.0 interrupt which occurs whenever High Speed Input pin 0 makes a zero-to-one transition. This interrupt will become pending in the INT\_PENDING register even if the HSI unit is programmed to ignore changes on HSI.0 or look for a one-to-zero transition.

### 3.7.5. Accessing Register Mapped I/O

The on-board I/O devices such as the serial port or the A/D converter are controlled as register mapped I/O. This allows convenient and efficient I/O processing. The implementation of the current members of the MCS-96 family place some restrictions on how these registers can be accessed. While these restrictions are not severe, the programmer must be aware of them. A complete listing of these registers is shown in figure 2-7 and 2-8. The restrictions are as follows:

- TIMER1, TIMER2 and HSI\_TIME are *word read only*. They cannot be read as bytes or written to in any format.
- HSO — TIME is *word write only*. It cannot be written to as individual bytes or read in any format.
- R0 (the ZERO register) is byte or word read or write but writing to it will not change its value.
- All of the other I/O registers can be accessed only as bytes. This applies even to the AD\_RESULT which is logically a word operand.



### 3.8. EXAMPLE-1 PROGRAMMING THE SERIAL I/O CHANNEL

MCS-96 MACRO ASSEMBLER SERIAL PORT DEMO PROGRAM

SERIES-III MCS-96 MACRO ASSEMBLER, V1.0

SOURCE FILE: :P1:SPX.SRC

OBJECT FILE: :P1:SPX.OBJ

CONTROLS SPECIFIED IN INVOCATION COMMAND: NOSB DEBUG

```

ERR LOC OBJECT LINE SOURCE STATEMENT
1 $TITLE('SERIAL PORT DEMO PROGRAM')
2 $PAGELENGTH(95)
3
4 ; This program initializes the serial port and echos any
5 ; character sent to it.
6
7
8 BAUD REG equ 0EH
9 SPCON equ 11H
10 SPSTAT equ 11H
11 IOCL equ 16H
12 IOC0 equ 15H
13 SBUF equ 07H
14 INT_PENDING equ 09H
15 SP equ 18H
16
17
18 rseg
19
20 CHR: dsb 1
21 TEMP0: dsb 1
22 TEMP1: dsb 1
23 RCV_FLAG: dsb 1
24
25
26 cseg
27
28 LD SP, #0B0H
29
30 LDB IOCL, #00100000B ; Set P2.0 to TXD
31
32 ; Baud rate = input frequency / (64*baud_val)
33 ; baud_val = (input frequency/64) / baud rate
34
35
36 baud_val equ 39 ; 2400 baud at 6.0 MHz
37
38 BAUD_HIGH equ ((baud_val-1)/256) OR 80H ; Set MSB to 1
39 BAUD_LOW equ (baud_val-1) MOD 256
40
41
42 LDB BAUD_REG, #BAUD_LOW
43 LDB BAUD_REG, #BAUD_HIGH
44
45 LDB SPCON, #01001001B ; Enable receiver, Mode 1
46
47 ; The serial port is now initialized
48
49
50 STB SBUF, CHR ; Clear serial Port
51 LDB TEMP0, #00100000B ; Set TI-temp
52
53 wait: JBC INT_PENDING, 6, wait ; Wait for pending bit to be set
54 ANDB INT_PENDING, #10111111B ; Clear pending bit
55
56 ORB TEMP0, SPCON ; Put SPCON into temp register
57 ; This is necessary because reading
58 ; SPCON clears TI and RI
59
60 get_byte:
61 JBC TEMP0, 6, put_byte ; If RI-temp is not set
62 STB SBUF, CHR ; Store byte
63 ANDB TEMP0, #10111111B ; CLR RI-temp
64 LDB RCV_FLAG, #0FFH ; Set bit-received flag
65
66 put_byte:
67 JBC RCV_FLAG, 0, continue ; If receive flag is cleared
68 JBC TEMP0, 5, continue ; If TI was not set
69 LDB SBUF, CHR ; Send byte
70 ANDB TEMP0, #10111111B ; CLR TI-temp
71 LDB RCV_FLAG, #00 ; Clear bit-received flag
72
73 continue:
74 BR wait
75
76 END

```

ASSEMBLY COMPLETED, NO ERROR(S) FOUND.

### 3.9. EXAMPLE-2 GENERATING A PWM WITH THE HSO UNIT

MCS-96 MACRO ASSEMBLER HSO EXAMPLE PROGRAM FOR PWM OUTPUTS

SERIES-III MCS-96 MACRO ASSEMBLER, V1.0

SOURCE FILE: :F1:HSO2X.SRC

OBJECT FILE: :F1:HSO2X.OBJ

CONTROLS SPECIFIED IN INVOCATION COMMAND: NOSB DEBUG

```

ERR LOC OBJECT LINE SOURCE STATEMENT
1 $TITLE ('HSO EXAMPLE PROGRAM FOR PWM OUTPUTS')
2 $PAGELENGTH (95)
3
4 ; This program will provide 4 PWM outputs on HSO pins 0-3
5 ; The input parameters passed to the program are:
6
7 ; HSO_ON_N HSO on time for pin N
8 ; HSO_OFF_N HSO off time for pin N
9
10 ; Where: Times are in timer1 cycles
11 ; N takes values from 0 to 3
12
13 ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
14
15 dseg
16
17
18 D_STAT: DSB 1
19 extrn HSO_ON_0:word, HSO_OFF_0:word
20 extrn HSO_ON_1:word, HSO_OFF_1:word
21 extrn HSO_ON_2:word, HSO_OFF_2:word
22 extrn HSO_ON_3:word, HSO_OFF_3:word
23 extrn HSO_TIME:word, HSO_COMMAND:byte
24 extrn TIMER1:word, IOS0:byte
25 extrn SP:word
26
27
28 rseg
29
30 public OLD_STAT
31 OLD_STAT: dsb 1
32 NEW_STAT: dsb 1
33
34
35 cseg
36
37 PUBLIC wait
38
39
40 wait: JBS IOS0, 6, wait ; Loop until HSO holding register
41 NOP ; is empty
42 NOP
43 STB IOS0, D_STAT ; Load byte to external RAM
44
45 ; For operation with interrupts 'store_stat:' would be the
46 ; entry point of the routine.
47 ; Note that a DI or PUSHF might have to be added.
48
49 store_stat:
50 ANDB NEW_STAT, IOS0, #0FH ; Store new status of HSO
51 CMPB OLD_STAT, NEW_STAT
52 JE wait ; If status hasn't changed
53 XORB OLD_STAT, NEW_STAT
54
55
56 check_0:
57 JBC OLD_STAT, 0, check_1 ; Jump if OLD_STAT(0)=NEW_STAT(0)
58 JBS NEW_STAT, 0, set_off_0
59
60 set_on_0:
61 LDB HSO_COMMAND, #00110000B ; Set HSO for timer1, set pin 0
62 ADD HSO_TIME, TIMER1, HSO_OFF_0 ; Time to set pin = Timer1 value
63 BR check_1 ; + Time for pin to be low
64
65 set_off_0:
66 LDB HSO_COMMAND, #00010000B ; Set HSO for timer1, clear pin 0
67 ADD HSO_TIME, TIMER1, HSO_ON_0 ; Time to clear pin = Timer1 value
68 ; + Time for pin to be high
69
70 check_1:
71 JBC OLD_STAT, 1, check_2 ; Jump if OLD_STAT(1)=NEW_STAT(1)
72 JBS NEW_STAT, 1, set_off_1
73
74 set_on_1:
75 LDB HSO_COMMAND, #00110001B ; Set HSO for timer1, set pin 1
76 ADD HSO_TIME, TIMER1, HSO_OFF_1 ; Time to set pin = Timer1 value
77 BR check_2 ; + Time for pin to be low
78
79 set_off_1:
80 LDB HSO_COMMAND, #00010001B ; Set HSO for timer1, clear pin 1
81 ADD HSO_TIME, TIMER1, HSO_ON_1 ; Time to clear pin = Timer1 value
82 ; + Time for pin to be high
83
84 $EJECT

```

# MCS®-96 SOFTWARE DESIGN INFORMATION

MCS-96 MACRO ASSEMBLER HSO EXAMPLE PROGRAM FOR PWM OUTPUTS

```

ERR LOC  OBJECT          LINE    SOURCE STATEMENT
      004A          86    check_2:
      004A 320017        R 87      JBC     OLD_STAT, 2, check_3      ; Jump if OLD_STAT(2)=NEW_STAT(2)
      004D 3A010B        R 88      JBS     NEW_STAT, 2, set_off_2
      0050          90    set_on_2:
      0050 B13200        E 91      LDB     HSO_COMMAND, #00110010B    ; Set HSO for timer1, set pin 2
      0053 470100000000  E 92      ADD     HSO_TIME, TIMER1, HSO_OFF_2  ; Time to set pin = Timer1 value
      0059 2009          93      BR      check_3                    ; + Time for pin to be low
      005B          95    set_off_2:
      005B B11200        E 96      LDB     HSO_COMMAND, #00010010B    ; Set HSO for timer1, clear pin 2
      005E 470100000000  E 97      ADD     HSO_TIME, TIMER1, HSO_ON_2    ; Time to clear pin = Timer1 value
      0059 2009          98      BR      check_3                    ; + Time for pin to be high
      0064          100    check_3:
      0064 330017        R 102     JBC     OLD_STAT, 3, check_done    ; Jump if OLD_STAT(3)=NEW_STAT(3)
      0067 3B010B        R 103     JBS     NEW_STAT, 3, set_off_3
      006A          104    set_on_3:
      006A B13300        E 106     LDB     HSO_COMMAND, #00110011B    ; Set HSO for timer1, set pin 3
      006D 470100000000  E 107     ADD     HSO_TIME, TIMER1, HSO_OFF_3  ; Time to set pin = Timer1 value
      0073 2009          108     BR      check_done                  ; + Time for pin to be low
      0075          110    set_off_3:
      0075 B11300        E 111     LDB     HSO_COMMAND, #00010011B    ; Set HSO for timer1, clear pin 3
      0078 470100000000  E 112     ADD     HSO_TIME, TIMER1, HSO_ON_3    ; Time to clear pin = Timer1 value
      0078 470100000000  E 113     BR      check_done                  ; + Time for pin to be high
      007E          116    check_done:
      007E B00100        R 117     LDB     OLD_STAT, NEW_STAT          ; Store current status and
      0081 F0          118     RET                                     ; wait for interrupt flag
      0081 F0          119     RET
      0082          120     END
      0082          121     END
      0082          122     END

```

ASSEMBLY COMPLETED, NO ERROR(S) FOUND.

### 3.10. EXAMPLE-3 MEASURING PULSES WITH THE HSI UNIT

MCS-96 MACRO ASSEMBLER MEASURING PULSES USING THE HSI UNIT

SERIES-III MCS-96 MACRO ASSEMBLER, V1.0

SOURCE FILE: :F1:PULSEX.SRC

OBJECT FILE: :F1:PULSEX.OBJ

CONTROLS SPECIFIED IN INVOCATION COMMAND: NOSB DEBUG

ERR LOC	OBJECT	LINE	SOURCE STATEMENT
		1	\$TITLE('MEASURING PULSES USING THE HSI UNIT')
		2	\$PAGELENGTH(95)
		3	
		4	; This program measures pulsewidths in TIMER1 cycles
		5	; and returns the values in external RAM.
		6	
		7	
0018		8	SP equ 18H
0003		9	HSI_MODE equ 03H
0006		10	HSI_STATUS equ 06H
0004		11	HSI_TIME equ 04H
000A		12	TIMER1 equ 0AH
0015		13	IOC0 equ 15H
0016		14	IOS1 equ 16H
		15	
		16	
0000		17	rseg
		18	
0000		19	HIGH_TIME: dsw 1
0002		20	LOW_TIME: dsw 1
0004		21	PERIOD: dsw 1
0006		22	HI_EDGE: dsw 1
0008		23	LO_EDGE: dsw 1
		24	
		25	
000A		26	AX: dsw 1
000A		27	AL equ AX :byte
000B		28	AH equ (AX+1) :byte
		29	
000C		30	BX: dsw 1
000C		31	BL equ BX :byte
000D		32	BU equ (BX+1) :byte
		33	; Note that 'BH' is an opcode so it
		34	; can't be used as a label
0000		35	cseg
		36	
0000	ALC00018	37	LD SP, #0C0H
0004	B10115	38	LDB IOC0, #00000001B ; Enable HSI 0
0007	B10F03	39	LDB HSI_MODE, #00001111B ; HSI 0 look for either edge
		40	
000A	44020004	41	wait: ADD PERIOD, HIGH_TIME, LOW_TIME
		42	
000E	3716F9	43	JBC IOS1, 7, wait ; Wait while no pulse is entered
		44	
0011	B0060A	45	LDB AL, HSI_STATUS ; Load status; Note that reading
		46	; HSI_TIME clears HSI_STATUS
		47	
0014	A0040C	48	LD BX, HSI_TIME ; Load the HSI_TIME
		49	
0017	390A09	50	JBS AL, 1, hsi_hi ; Jump if HSI.0 is high
		51	
001A	C0080C	52	hsi_lo: ST BX, LO_EDGE
001D	48060800	53	SUB HIGH_TIME, LO_EDGE, HI_EDGE
0021	27E7	54	BR wait
		55	
		56	
0023	C0060C	57	hsi_hi: ST BX, HI_EDGE
0026	48080602	58	SUB LOW_TIME, HI_EDGE, LO_EDGE
002A	27DE	59	BR wait
		60	
002C		61	END

ASSEMBLY COMPLETED, NO ERROR(S) FOUND.



## 3.11. EXAMPLE-4 SCANNING THE A/D CHANNELS

MCS-96 MACRO ASSEMBLER SCANNING THE A TO D CHANNELS

SERIES-III MCS-96 MACRO ASSEMBLER, V1.0

SOURCE FILE: :F1:ATODX.SRC

OBJECT FILE: :F1:ATODX.OBJ

CONTROLS SPECIFIED IN INVOCATION COMMAND: NOSB DEBUG

```

ERR LOC  OBJECT          LINE    SOURCE STATEMENT
1 $TITLE('SCANNING THE A TO D CHANNELS')
2 $PAGELENGTH(95)
3
4 ; This program scans A to D lines 0 through 3 and stores the
5 ; results in RESULT_N
6
7 AD_RESULT_LO equ 02
8 AD_RESULT_HI equ 03
9 AD_COMMAND equ 02
10 SP equ 18H
11
12
13 dseg
14
15 RESULT_TABLE:
16 RESULT 1: dsw 1
17 RESULT 2: dsw 1
18 RESULT 3: dsw 1
19 RESULT 4: dsw 1
20
21 rseg
22
23 AX: dsw 1
24 AL equ AX :byte
25 AH equ (AX+1) :byte
26
27 BX: dsw 1
28 BL equ BX :byte
29 BU equ (BX+1) :byte ; Note that 'BH' is an opcode so it
30 ; can't be used as a label
31
32
33 DX: dsw 1
34 DL equ DX :byte
35 DH equ (DX+1) :byte
36
37
38 cseg
39
40
41 start: LD SP, #0C0H ; Set Stack Pointer
42 LD BX, 00H ; Use the zero register
43
44 next: ORB BL, #1000B ; Start conversion on channel
45 LDB AD_COMMAND, BL ; indicated by BL register
46 ANDB BL, #0111B
47
48 NOP ; Wait for conversion to start
49
50 check: JBS AD_RESULT_LO, 3, check ; Wait while A to D is busy
51
52
53
54
55 LDB AL, AD_RESULT_LO ; Load low order result
56 LDB AH, AD_RESULT_HI ; Load high order result
57
58
59 ADDB DL, BL, BL ; DL=BL*2
60 LDBZ DX, DL
61 ST AX, RESULT_TABLE[DX] ; Store result indexed by BL*2
62
63
64 INCB BL
65 ANDB BL, #03H
66
67 BR next
68
69 END

```

ASSEMBLY COMPLETED, NO ERROR(S) FOUND.

## 3.12. EXAMPLE-5 TABLE LOOKUP-AND INTERPOLATION

MCS-96 MACRO ASSEMBLER TABLE LOOKUP AND INTERPOLATION

SERIES-III MCS-96 MACRO ASSEMBLER, V1.0

SOURCE FILE: :F1:INTERX.SRC

OBJECT FILE: :F1:INTERX.OBJ

CONTROLS SPECIFIED IN INVOCATION COMMAND: NOSB DEBUG

```

ERR LOC OBJECT          LINE SOURCE STATEMENT
1  $TITLE('TABLE LOOKUP AND INTERPOLATION')
2  $PAGELENGTH(95)
3
4  ; This program uses a lookup table to generate 12-bit function values
5  ; using 8-bit input values. The table is 16 bytes long and 16 bits wide.
6  ; A linear interpolation is made using the following formula:
7
8  ; Table Step      IN_VAL - TABLE_LOW
9  ; ----- = -----
10 ; TABLE_HIGH - TABLE_LOW  OUT - TABLE_LOW
11 ;
12 ; 16      IN_DIF
13 ; ----- = -----
14 ; TAB_DIF  OUT_DIF
15 ;
16 ; Cross Multiplication is used to solve for OUT_DIF
17
18
19
20      SP      equ      18H
21
22
23      dseg
24
25      RESULT_TABLE:
26      RESULT:      dsw      1
27
28
29      rseg
30
31      AX:      dsw      1
32      AL      equ      AX      :byte
33      AH      equ      (AX+1)  :byte
34
35      BX:      dsw      1
36      BL      equ      BX      :byte
37      BU      equ      (BX+1)  :byte      ; Note that 'BH' is an opcode so it
38                                          ; can't be used as a label
39
40      IN_VAL:      dsb      1
41      TABLE_LOW:  dsw      1
42      TABLE_HIGH: dsw      1
43      IN_DIF:      dsw      1
44      IN_DIFB:      equ     IN_DIF :byte
45      TAB_DIF:      dsw      1
46      OUT_DIF:      dsw      1
47      OUT_DIFB:     dsl      1
48
49
50      cseg
51
52
53      start: LD      SP, #0C0H      ; Set Stack Pointer
54
55
56      look: LDB      AL, IN_VAL
57      SHRB      AL, #3              ; Place 2 times the upper nibble in byte
58      ANDB      AL, #11111110B     ; Insure AL is a word address
59      LDBZE     AX, AL
60
61      LD      TABLE_LOW, TABLE [AX] ; TABLE LOW is table output value
62                                          ; of IN_VAL rounded down to the
63                                          ; nearest multiple of 10H.
64
65      LD      TABLE_HIGH, (TABLE+2) [AX] ; TABLE HIGH is the table output
66                                          ; value of IN_VAL rounded up to the
67                                          ; nearest multiple of 10H.
68
69
70      SUB      TAB_DIF, TABLE_HIGH, TABLE_LOW
71
72      ANDB      IN_DIFB, IN_VAL, #0FH
73      LDBSE     IN_DIF, IN_DIFB      ; Make input difference into a word
74
75      MUL      OUT_DIF, IN_DIF, TAB_DIF
76      DIV      OUT_DIF, #16
77
78      labl: ADD      OUT, OUT_DIF, TABLE_LOW ; Add output difference to output
79                                          ; generated with truncated IN_VAL
80                                          ; as input
81      SHR      OUT, #4              ; Round to 12-bit answer

```

# MCS®-96 SOFTWARE DESIGN INFORMATION

## MCS-96 MACRO ASSEMBLER

## TABLE LOOKUP AND INTERPOLATION

10/11/83  
3.12. EXAMINATION  
INTERPOLATION

ERR LOC OBJECT  
0036 D307  
0038 070E  
003A C3010000E

R  
R

LINE SOURCE STATEMENT  
82 JNC lab2  
83 INC OUT  
84 ST OUT, RESULT

; Round up if Carry = 1

003F 27C3

86 lab2: BR look

0041

87  
88 cseg  
89  
90

0042 000000200034004C  
004A 005D006A00720078  
0052 007B007D0076006D  
005A 005D004B00340022  
0062 0010

91 table: DCW 0000H, 2000H, 3400H, 4C00H  
92 DCW 5D00H, 6A00H, 7200H, 7800H  
93 DCW 7B00H, 7D00H, 7600H, 6D00H  
94 DCW 5D00H, 4B00H, 3400H, 2200H  
95 DCW 1000H

; A random non-monotonic  
; function

0064 97 END  
ASSEMBLY COMPLETED, NO ERROR(S) FOUND.

### 3.13. DETAILED INSTRUCTION SET DESCRIPTION

This section gives a description of each instruction recognized by the 8096 sorted alphabetically by the mnemonic used in the assembly language for the 8096. Note that the effect on the program counter (PC) is not always shown in the instruction descriptions. All instructions increment the PC by the number of bytes in the instruction. Several acronyms are used in the instruction set descriptions which are defined here:

**aa.** A two bit field within an opcode which selects the basic addressing mode user. This field is only present in those opcodes which allow address mode options. The encoding of the field is as follows:

aa	Addressing mode
00	Register direct
01	Immediate
10	Indirect
11	Indexed

The selection between indirect and indirect with auto-increment or between short and long indexing is done based on the least significant bit of the instruction byte which follows the opcode. This type selects the 16-bit register which is to take part in the address calculation. Since the 8096 requires that words be aligned on even byte boundaries this bit would be otherwise unused.

**breg.** A byte register in the internal register file. When confusion could exist as to whether this field refers to a source or a destination register it will be prefixed with an "S" or a "D."

**baop.** A byte operand which is addressed by any of the address modes discussed in section 3.2.

**bitno.** A three bit field within an instruction op-code which selects one of the eight bits in a byte.

**wreg.** A word register in the internal register file. When confusion could exist as to whether this field refers to a source register or a destination register it will be prefixed with an "S" or a "D."

**waop.** A word operand which is addressed by any of the address modes discussed in section 3.2.

**Lreg.** A 32-bit register in the internal register file.

**BEA.** Extra bytes of code required for the address mode selected.

**CEA.** Extra state times (cycles) required for the address mode selected.

**cadd** An address in the program code.

**Flag Settings.** The modification to the flag setting is shown for each instruction. A checkmark (✓) means that the flag is set or cleared as appropriate. A hyphen means that the flag is not modified. A one or zero (1) or (0) indicates that the flag will be in that state after the instruction. An up arrow (↑) indicates that the instruction may set the flag if it is appropriate but will not clear the flag. A down arrow (↓) indicates that the flag can be cleared but not set by the instruction. A question mark (?) indicates that the flag will be left in an indeterminate state after the operation.

**Generic Jumps and Calls.** The assembler for the 8096 provides for generic jumps and calls. For all of the conditional jump instructions a "B" can be substituted for the "J" and the assembler will generate a code sequence which is logically equivalent but can reach anywhere in the memory. A JH can only jump about 128 locations from the current program counter; a BH can jump anywhere in memory. In a like manner a BR will cause a SJMP or LJMP to be generated as appropriate and a CALL will cause a SCALL or LCALL to be generated. The assembler user guide (see section 3.0) should be consulted for the algorithms used by the assembler to convert these generic instructions into actual machine instructions.



## 3.13.1. ADD (Two Operands) — ADD WORDS

**Operation:** The sum of the two word operands is stored into the destination (leftmost) operand.

$$(DEST) \leftarrow (DEST) + (SRC)$$

**Assembly Language Format:**

	DST	SRC
ADD	wreg,	waop

**Object Code Format:** [ 011001aa ][ waop ][ wreg ]

Bytes: 2+BEA

States: 4+CEA

Flags Affected

Z	N	C	V	VT	ST
✓	✓	✓	✓	↑	-

## 3.13.2. ADD (Three Operands) — ADD WORDS

**Operation:** The sum of the second and third word operands is stored into the destination (leftmost) operand.

$$(DEST) \leftarrow (SRC1) + (SRC2)$$

**Assembly Language Format:**

	DST	SRC1	SRC2
ADD	Dwreg, Swreg,	waop	

**Object Code Format:** [ 010001aa ][ waop ][ Swreg ][ Dwreg ]

Bytes: 3+BEA

States: 5+CEA

Flags Affected

Z	N	C	V	VT	ST
✓	✓	✓	✓	↑	-

### 3.13.3. ADDDB (Two Operands) — ADD BYTES

**Operation:** The sum of the two byte operands is stored into the destination (leftmost) operand.

$$(DEST) \leftarrow (DEST) + (SRC)$$

**Assembly Language Format:**

DST SRC  
ADDB breg, baop

**Object Code Format:** [ 011101aa ][ baop ][ breg ]

Bytes: 2 + BEA

States: 4 + CEA

Flags Affected					
Z	N	C	V	VT	ST
✓	✓	✓	✓	↑	—

### 3.13.4. ADDDB (Three Operands) — ADD BYTES

**Operation:** The sum of the second and third byte operands is stored into the destination (leftmost) operand.

$$(DEST) \leftarrow (SRC1) + (SRC2)$$

**Assembly Language Format:**

DST SRC1 SRC2  
ADDB Dbreg, Sbreg, baop

**Object Code Format:** [ 010101aa ][ baop ][ Sbreg ][ Dbreg ]

Bytes: 3 + BEA

States: 5 + CEA

Flags Affected					
Z	N	C	V	VT	ST
✓	✓	✓	✓	↑	—

**3.13.5. ADDC — ADD WORDS WITH CARRY**

**Operation:** The sum of the two word operands and the carry flag (0 or 1) is stored into the destination (leftmost) operand.

$$(DEST) \leftarrow (DEST) + (SRC) + C$$

**Assembly Language Format:**

DST SRC  
ADDC wreg, waop

**Object Code Format:** [ 101001aa ] [ waop ] [ wreg ]

Bytes: 2 + BEA

States: 4 + BEA

Flags Affected					
Z	N	C	V	VT	ST
↓	✓	✓	✓	↑	—

**3.13.6. ADDCB — ADD BYTES WITH CARRY**

**Operation:** The sum of the two byte operands and the carry flag (0 or 1) is stored into the destination (leftmost) operand.

$$(DEST) \leftarrow (DEST) + (SRC) + C$$

**Assembly Language Format:**

DST SRC  
ADDCB breg, baop

**Object Code Format:** [ 101101aa ] [ baop ] [ breg ]

Bytes: 2 + BEA

States: 4 + CEA

Flags Affected					
Z	N	C	V	VT	ST
↓	✓	✓	✓	↑	—

**3.13.7. AND (Two Operands) — LOGICAL AND WORDS**

**Operation:** The two word operands are ANDed, the result having a 1 only in those bit positions where both operands had a 1, with zeroes in all other bit positions. The result is stored into the destination (leftmost) operand.

$(\text{DEST}) \leftarrow (\text{DEST}) \text{ AND } (\text{SRC})$

**Assembly Language Format:**

DST SRC  
AND wreg, waop

**Object Code Format:** [ 011000aa ][ waop ][ wreg ]

Bytes: 2 + BEA

States: 4 + CEA

Flags Affected					
Z	N	C	V	VT	ST
✓	✓	0	0	—	—

**3.13.8. AND (Three Operands) — LOGICAL AND WORDS**

**Operation:** The second and third word operands are ANDed, the result having a 1 only in those bit positions where both operands had a 1, with zeroes in all other bit positions. The result is stored into the destination (leftmost) operand.

$(\text{DEST}) \leftarrow (\text{SRC1}) \text{ AND } (\text{SRC2})$

**Assembly Language Format:**

DST SRC1 SRC2  
AND Dwreg, Swreg, waop

**Object Code Format:** [ 010000aa ][ waop ][ Swreg ][ Dwreg ]

Bytes: 3 + BEA

States: 5 + CEA

Flags Affected					
Z	N	C	V	VT	ST
✓	✓	0	0	—	—



**3.13.9. ANDB (Two Operands) — LOGICAL AND BYTES**

**Operation:** The two byte operands are ANDed, the result having a 1 only in those bit positions where both operands had a 1, with zeroes in all other bit positions. The result is stored into the destination (leftmost) operand.

$$(DEST) \leftarrow (DEST) \text{ AND } (SRC)$$

**Assembly Language Format:**

DST SRC  
ANDB breg, baop

**Object Code Format:** [ 011100aa ][ baop ][ breg ]

Bytes: 2 + BEA

States: 4 + CEA

Flags Affected					
Z	N	C	V	VT	ST
✓	✓	0	0	—	—

**3.13.10. ANDB (Three Operands) — LOGICAL AND BYTES**

**Operation:** The second and third byte operands are ANDed, the result having a 1 only in those bit positions where both operands had a 1, with zeroes in all other bit positions. The result is stored into the destination (leftmost) operand.

$$(DEST) \leftarrow (SRC1) \text{ AND } (SRC2)$$

**Assembly Language Format:**

DST SRC1 SRC2  
ANDB Dbreg, Sbreg, baop

**Object Code Format:** [ 010100aa ][ baop ][ Sbreg ][ Dbreg ]

Bytes: 3 + BEA

States: 5 + CEA

Flags Affected					
Z	N	C	V	VT	ST
✓	✓	0	0	—	—

**3.13.11. BR (Indirect) — BRANCH INDIRECT**

**Operation:** The execution continues at the address specified in the operand word register.

$PC \leftarrow (DEST)$

**Assembly Language Format:** BR wreg

**Object Code Format:** [ 11100011 ][wreg]

Bytes: 2

States: 8

Flags Affected					
Z	N	C	V	VT	ST
—	—	—	—	—	—

**3.13.12. CLR — CLEAR WORD**

**Operation:** The value of the word operand is set to zero.

$(DEST) \leftarrow 0$

**Assembly Language Format:** CLR wreg

**Object Code Format:** [ 00000001 ][ wreg ]

Bytes: 2

States: 4

Flags Affected					
Z	N	C	V	VT	ST
1	0	0	0	—	—

**3.13.13. CLRB — CLEAR BYTE**

**Operation:** The value of the byte operand is set to zero.

$(\text{DEST}) \leftarrow 0$

**Assembly Language Format:** CLRB breg

**Object Code Format:** [ 00010001 ][ breg ]

Bytes: 2

States: 4

Flags Affected					
Z	N	C	V	VT	ST
1	0	0	0	-	-

**3.13.14. CLRC — CLEAR CARRY FLAG**

**Operation:** The value of the carry flag is set to zero.

$C \leftarrow 0$

**Assembly Language Format:** CLRC

**Object Code Format:** [ 11111000 ]

Bytes: 1

States: 4

Flags Affected					
Z	N	C	V	VT	ST
-	-	0	-	-	-

### 3.13.15. CLRVT — CLEAR OVERFLOW TRAP

**Operation:** The value of the overflow-trap flag is set to zero.

$VT \leftarrow 0$

**Assembly Language Format:** CLRVT

**Object Code Format:** [ 11111100 ]

Bytes: 1

States: 4

Flags Affected					
Z	N	C	V	VT	ST
-	-	-	-	0	-

### 3.13.16. CMP — COMPARE WORDS

**Operation:** The source (rightmost) word operand is subtracted from the destination (leftmost) word operand. The flags are altered but the operands remain unaffected. The carry flag is set as complement of borrow.

(DEST) — (SRC)

**Assembly Language Format:** DST SRC

CMP wreg, waop

**Object Code Format:** [ 100010aa ][ waop ][ wreg ]

Bytes: 2+BEA

States: 4+CEA

Flags Affected					
Z	N	C	V	VT	ST
✓	✓	✓	✓	↑	-



**3.13.17. CMPB — COMPARE BYTES**

**Operation:** The source (rightmost) byte operand is subtracted from the destination (leftmost) byte operand. The flags are altered but the operands remain unaffected. The carry flag is set as complement of borrow.

(DEST) — (SRC)

**Assembly Language Format:**

DST SRC  
CMPB breg, baop

**Object Code Format:** [ 100110aa ][ baop ][ breg ]

Bytes: 2 + BEA

States: 4 + CEA

Flags Affected					
Z	N	C	V	VT	ST
✓	✓	✓	✓	↑	—

**3.13.18. DEC — DECREMENT WORD**

**Operation:** The value of the word operand is decremented by one.

(DEST) ← (DEST) — 1

**Assembly Language Format:**

DEC wreg<sub>w</sub>

**Object Code Format:** [ 00000101 ][ wreg ]

Bytes: 2

States: 4

Flags Affected					
Z	N	C	V	VT	ST
✓	✓	✓	✓	↑	—

## 3.13.19. DECB — DECREMENT BYTE

**Operation:** The value of the byte operand is decremented by one.

$(\text{DEST}) \leftarrow (\text{DEST}) - 1$

**Assembly Language Format:** DECB breg

**Object Code Format:** [ 00010101 ] [ breg ]

Bytes: 2

States: 4

Flags Affected					
Z	N	C	V	VT	ST
✓	✓	✓	✓	↑	—

## 3.13.20. DI — DISABLE INTERRUPTS

**Operation:** Interrupts are disabled. Interrupt-calls will not occur after this instruction.

Interrupt Enable (PSW.9)  $\leftarrow 0$

**Assembly Language Format:** DI

**Object Code Format:** [ 11111010 ]

Bytes: 1

States: 4

Flags Affected					
Z	N	C	V	VT	ST
—	—	—	—	—	—

## 3.13.21. DIV — DIVIDE INTEGERS

**Operation:** This instruction divides the contents of the destination LONG-INTEGER operand by the contents of the INTEGER word operand, using signed arithmetic. The low order word of the destination (i.e., the word with the lower address) will contain the quotient; the high order word will contain the remainder.

(low word DEST)  $\leftarrow$  (DEST) / (SRC)  
 (high word DEST)  $\leftarrow$  (DEST) MOD (SRC)

The above two statements are performed concurrently.

**Assembly Language Format:**            DST     SRC  
 DIV   lreg,     waop

**Object Code Format:** [ 11111110 ][ 100011aa ][ waop ][ lreg ]

Bytes: 2 + BEA  
 States: 29 + CEA

Flags Affected					
Z	N	C	V	VT	ST
—	—	—	✓	↑	—

## 3.13.22. DIVB — DIVIDE SHORT-INTEGERS

**Operation:** This instruction divides the contents of the destination INTEGER operand by the contents of the source SHORT-INTEGER operand, using signed arithmetic. The low order byte of the destination (i.e. the byte with the lower address) will contain the quotient; the high order byte will contain the remainder.

(low byte DEST)  $\leftarrow$  (DEST) / (SRC)  
 (high byte DEST)  $\leftarrow$  (DEST) MOD (SRC)

The above two statements are performed concurrently.

**Assembly Language Format:**            DST     SRC  
 DIVB   wreg,     baop

**Object Code Format:** [ 11111110 ][ 100111aa ][ baop ][ wreg ]

Bytes: 2 + BEA  
 States: 21 + CEA

Flags Affected					
Z	N	C	V	VT	ST
—	—	—	✓	↑	—

**3.13.23. DIVU — DIVIDE WORDS**

**Operation:** This instruction divides the contents of the destination DOUBLE-WORD operand by the contents of the source WORD operand, using unsigned arithmetic. The low order word will contain the quotient; the high order byte will contain the remainder.

(low word DEST)  $\leftarrow$  (DEST) / (SRC)

(high word DEST)  $\leftarrow$  MOD (SRC)

The above two statements are performed concurrently.

**Assembly Language Format:**           DST     SRC  
DIVU   lreg,   waop

**Object Code Format:** [ 100011aa ][ waop ][ lreg ]

Bytes: 2 + BEA

States: 25 + CEA

Flags Affected					
Z	N	C	V	VT	ST
-	-	-	✓	↑	-

Flags Affected					
Z	N	C	V	VT	ST
-	-	-	-	-	-

**3.13.24. DIVUB — DIVIDE BYTES**

**Operation:** This instruction divides the contents of the destination WORD operand by the contents of the source BYTE operand, using unsigned arithmetic. The low order byte of the destination (i.e., the byte with the lower address) will contain the quotient; the high order byte will contain the remainder.

(low byte DEST)  $\leftarrow$  (DEST) / (SRC)

(high byte DEST)  $\leftarrow$  (DEST) MOD (SRC)

The above two statements are performed concurrently.

**Assembly Language Format:**           DST     SRC  
DIVUB   wreg,   baop

**Object Code Format:** [ 100111aa ][ baop ][ wreg ]

Bytes: 2 + BEA

States: 17 + CEA

Flags Affected					
Z	N	C	V	VT	ST
-	-	-	✓	↑	-



**3.13.25. DJNZ — DECREMENT AND JUMP IF NOT ZERO**

**Operation:** The value of the byte operand is decremented by 1. If the result is not equal to 0, the distance from the end of this instruction to the target label is added to the program counter, effecting the jump. The offset from the end of this instruction to the target label must be in the range of -128 to +127. If the result of the decrement is zero then control passes to the next sequential instruction.

$(COUNT) \leftarrow (COUNT) - 1$

if  $(COUNT) < > 0$  then

PC  $\leftarrow$  PC + disp (sign-extended to 16 bits)

end \_ if

**Assembly Language Format:** DJNZ breg,cadd

**Object Code Format:** [ 11100000 ][ breg ][ disp ]

Bytes: 3

States: Jump Not Taken: 5

Jump Taken: 9

Flags Affected					
Z	N	C	V	VT	ST
-	-	-	-	-	-

**3.13.26. EI — ENABLE INTERRUPTS**

**Operation:** Interrupts are enabled following the execution of the next statement. Interrupt-calls cannot occur immediately following this instruction.

Interrupt Enable (PSW.9)  $\leftarrow$  1

**Assembly Language Format:** EI

**Object Code Format:** [ 11111011 ]

Bytes: 1

States: 4

Flags Affected					
Z	N	C	V	VT	ST
-	-	-	-	-	-

**3.13.27. EXT — SIGN EXTEND INTEGER INTO LONG-INTEGER**

**Operation:** The low order word of the operand is sign-extended throughout the high order word of the operand.

if (low word DEST) < 8000H then  
 (high word DEST) ← 0  
 else  
 (high word DEST) ← 0FFFFH  
 end \_ if

**Assembly Language Format:** EXT lreg

**Object Code Format:** [ 00000110 ][ lreg ]

Bytes: 2

States: 4

Flags Affected					
Z	N	C	V	VT	ST
✓	✓	0	0	—	—

**3.13.28. EXTB — SIGN EXTEND SHORT-INTEGER INTO INTEGER**

**Operation:** The low order byte of the operand is sign-extended throughout the high order byte of the operand.

if (low byte DEST) < 80H then  
 (high byte DEST) ← 0  
 else  
 (high byte DEST) ← 0FFH  
 end \_ if

**Assembly Language Format:** EXTB wreg

**Object Code Format:** [ 00010110 ][ wreg ]

Bytes: 2

States: 4

Flags Affected					
Z	N	C	V	VT	ST
✓	✓	0	0	—	—

**3.13.29. INC — INCREMENT WORD****Operation:** The value of the word operand is incremented by 1.

$$(\text{DEST}) \leftarrow (\text{DEST}) + 1$$

**Assembly Language Format:** INC wreg**Object Code Format:** [ 00000111 ][ wreg ]

Bytes: 2

States: 4

Flags Affected					
Z	N	C	V	VT	ST
✓	✓	✓	✓	↑	—

Flags Affected					
Z	N	C	V	VT	ST
✓	✓	✓	✓	↑	—

**3.13.30. INCB — INCREMENT BYTE****Operation:** The value of the byte operand is incremented by 1.

$$(\text{DEST}) \leftarrow (\text{DEST}) + 1$$

**Assembly Language Format:** INCB breg**Object Code Format:** [ 00010111 ][ breg ]

Bytes: 2

States: 4

Flags Affected					
Z	N	C	V	VT	ST
✓	✓	✓	✓	↑	—

Flags Affected					
Z	N	C	V	VT	ST
✓	✓	✓	✓	↑	—

**Operation:** The specified bit is tested. If it is clear (i.e., 0), the distance from the end of this instruction to the target label is added to the program counter, effecting the jump. The offset from the end of this instruction to the target label must be in the range of  $-128$  to  $+127$ . If the bit is set (i.e., 1), control passes to the next sequential instruction.

if (specified bit)  $\neq 0$  then

$PC \leftarrow PC + \text{disp}$  (sign-extended to 16 bits)

**Assembly Language Format:** JBC breg,bitno,cadd

**Object Code Format:** [ 00110bbb ] [ breg ] [ disp ]

where bbb is the bit number within the specified register.

Bytes: 3

States: Jump Not Taken: 5

Jump Taken: 9

Flags Affected					
Z	N	C	V	VT	ST
-	-	-	-	-	-

## 3.13.33. JC — JUMP IF CARRY FLAG IS SET

**Operation:** If the carry flag is set (i.e., 1), the distance from the end of this instruction to the target label is added to the program counter, effecting the jump. The offset from the end of this instruction to the target label must be in the range of  $-128$  to  $+127$ . If the carry flag is clear (i.e., 0), control passes to the next sequential instruction.

if C = 1 then

$PC \leftarrow PC + \text{disp}$  (sign-extended to 16 bits)

**Assembly Language Format:** JC cadd

**Object Code Format:** [ 11011011 ] [ disp ]

Bytes: 2

States: Jump Not Taken: 4

Jump Taken: 8

Flags Affected					
Z	N	C	V	VT	ST
-	-	-	-	-	-



## 3.13.32. JBS — JUMP IF BIT SET

**Operation:** The specified bit is tested. If it is set (i.e., 1), the distance from the end of this instruction to the target label is added to the program counter, effecting the jump. The offset from the end of this instruction to the target label must be in the range of -128 to +127. If the bit is clear (i.e., 0), control passes to the next sequential instruction.

if (specified bit) = 1 then  
 $PC \leftarrow PC + \text{disp}$  (sign-extended to 16 bits)

**Assembly Language Format:** JBS breg,bitno,cadd

**Object Code Format:** [ 00111bbb ][ breg ][ disp ]

where bbb is the bit number within the specified register.

Bytes: 3  
 States: Jump Not Taken: 5  
 Jump Taken: 9

Flags Affected					
Z	N	C	V	VT	ST
-	-	-	-	-	-

## 3.13.33. JC — JUMP IF CARRY FLAG IS SET

**Operation:** If the carry flag is set (i.e., 1), the distance from the end of this instruction to the target label is added to the program counter, effecting the jump. The offset from the end of this instruction to the target label must be in the range of -128 to +127. If the carry flag is clear (i.e., 0), control passes to the next sequential instruction.

if C = 1 then  
 $PC \leftarrow PC + \text{disp}$  (sign-extended to 16 bits)

**Assembly Language Format:** JC cadd

**Object Code Format:** [ 11011011 ][ disp ]

Bytes: 2  
 States: Jump Not Taken: 4  
 Jump Taken: 8

Flags Affected					
Z	N	C	V	VT	ST
-	-	-	-	-	-

**3.13.34. JE — JUMP IF EQUAL**

**Operation:** If the zero flag is set (i.e., 1), the distance from the end of this instruction to the target label is added to the program counter, effecting the jump. The offset from the end of this instruction to the target label must be in the range of -128 to +127. If the zero flag is clear (i.e., 0), control passes to the next sequential instruction.

if Z = 1 then  
 $PC \leftarrow PC + \text{disp (sign-extended to 16 bits)}$

**Assembly Language Format:** JE cadd

**Object Code Format:** [ 11011111 ][ disp ]

Bytes: 2  
 States: Jump Not Taken: 4  
 Jump Taken: 8

Flags Affected					
Z	N	C	V	VT	ST
-	-	-	-	-	-

**3.13.35. JGE — JUMP IF SIGNED GREATER THAN OR EQUAL**

**Operation:** If the negative flag is clear (i.e., 0), the distance from the end of this instruction to the target label is added to the program counter, effecting the jump. The offset from the end of this instruction to the target label must be in the range of -128 to +127. If the negative flag is set (i.e., 1), control passes to the next sequential instruction.

if N = 0 then  
 $PC \leftarrow PC + \text{disp (sign-extended to 16 bits)}$

**Assembly Language Format:** JGE cadd

**Object Code Format:** [ 11010110 ][ disp ]

Bytes: 2  
 States: Jump Not Taken: 4  
 Jump Taken: 8

Flags Affected					
Z	N	C	V	VT	ST
-	-	-	-	-	-

**3.13.36. JGT — JUMP IF SIGNED GREATER THAN**

**Operation:** If both the negative flag and the zero flag are clear (i.e., 0), the distance from the end of this instruction to the target label is added to the program counter, effecting the jump. The offset from the end of this instruction to the target label must be in the range of  $-128$  to  $+127$ . If either the negative flag or the zero flag are set (i.e., 1,) control passes to the next sequential instruction.

if  $N = 0$  AND  $Z = 0$  then  
 $PC \leftarrow PC + \text{disp}$  (sign-extended to 16 bits)

**Assembly Language Format:** JGT cadd

**Object Code Format:** [ 11010010 ][ disp ]

Bytes: 2  
 States: Jump Not Taken: 4  
 Jump Taken: 8

Flags Affected					
Z	N	C	V	VT	ST
-	-	-	-	-	-

**3.13.37. JH — JUMP IF HIGHER (UNSIGNED)**

**Operation:** If the carry flag is set (i.e., 1), but the zero flag is not, the distance from the end of this instruction to the target label is added to the program counter, effecting the jump. The offset from the end of this instruction to the target label must be in the range of  $-128$  to  $+127$ . If either the carry flag is clear or the zero flag is set, control passes to the next sequential instruction.

if  $C = 1$  and  $Z = 0$  then  
 $PC \leftarrow PC + \text{disp}$  (sign-extended to 16 bits)

**Assembly Language Format:** JH cadd

**Object Code Format:** [ 11011001 ][ disp ]

Bytes: 2  
 States: Jump Not Taken: 4  
 Jump Taken: 8

Flags Affected					
Z	N	C	V	VT	ST
-	-	-	-	-	-

**3.13.38. JLE — JUMP IF SIGNED LESS THAN OR EQUAL**

**Operation:** If either the negative flag or the zero flag are set (i.e., 1), the distance from the end of this instruction to the target label is added to the program counter, effecting the jump. The offset from the end of this instruction to the target label must be in the range of -128 to +127. If both the negative flag and the zero flag are clear (i.e., 0), control passes to the next sequential instruction.

if  $N = 1$  OR  $Z = 1$  then

$PC \leftarrow PC + \text{disp}$  (sign-extended to 16 bits)

**Assembly Language Format:** JLE cadd

**Object Code Format:** [ 11011010 ][ disp ]

Bytes: 2

States: Jump Not Taken: 4

Jump Taken: 8

Flags Affected					
Z	N	C	V	VT	ST
-	-	-	-	-	-

**3.13.39. JLT — JUMP IF SIGNED LESS THAN**

**Operation:** If the negative flag is set (i.e., 1), the distance from the end of this instruction to the target label is added to the program counter, effecting the jump. The offset from the end of this instruction to the target label must be in the range of -128 to +127. If the negative flag is clear (i.e., 0), control passes to the next sequential instruction.

if  $N = 1$  then

$PC \leftarrow PC + \text{disp}$  (sign-extended to 16 bits)

**Assembly Language Format:** JLT cadd

**Object Code Format:** [ 11011110 ][ disp ]

Bytes: 2

States: Jump Not Taken: 4

Jump Taken: 8

Flags Affected					
Z	N	C	V	VT	ST
-	-	-	-	-	-



**Operation:** If the carry flag is clear (i.e., 0), the distance from the end of this instruction to the target label is added to the program counter, effecting the jump. The offset from the end of this instruction to the target label must be in the range of -128 to +127. If the carry flag is set (i.e., 1), control passes to the next sequential instruction.

if C = 0 then  
 $PC \leftarrow PC + \text{disp}$  (sign-extended to 16 bits)

**Assembly Language Format:** JNC cadd

**Object Code Format:** [ 11010011 ] [ disp ]

Bytes: 2  
 States: Jump Not Taken: 4  
 Jump Taken: 8

Flags Affected					
Z	N	C	V	VT	ST
-	-	-	-	-	-

### 3.13.41. JNE — JUMP IF NOT EQUAL

**Operation:** If the zero flag is clear (i.e., 0), the distance from the end of this instruction to the target label is added to the program counter, effecting the jump. The offset from the end of this instruction to the target label must be in the range of -128 to +127. If the zero flag is set (i.e., 1), control passes to the next sequential instruction.

if Z = 0 then  
 $PC \leftarrow PC + \text{disp}$  (sign-extended to 16 bits)

**Assembly Language Format:** JNE cadd

**Object Code Format:** [ 11010111 ] [ disp ]

Bytes: 2  
 States: Jump Not Taken: 4  
 Jump Taken: 8

Flags Affected					
Z	N	C	V	VT	ST
-	-	-	-	-	-

**3.13.42. JNH — JUMP IF NOT HIGHER (UNSIGNED)**

**Operation:** If either the carry flag is clear (i.e., 0), or the zero flag is set (i.e., 1), the distance from the end of this instruction to the target label is added to program counter, effecting the jump. The offset from the end of this instruction to the target label must be in the range of -128 to +127. If the carry flag is set (i.e., 1), or the zero flag is not, control passes to the next sequential instruction.

if  $C = 0$  OR  $Z = 1$  then

$PC \leftarrow PC + \text{disp}$  (sign-extended to 16 bits)

**Assembly Language Format:** JNH cadd

**Object Code Format:** [ 11010001 ][ disp ]

Bytes: 2

States: Jump Not Taken: 4

Jump Taken: 8

Flags Affected

Z	N	C	V	VT	ST
-	-	-	-	-	-

**3.13.43. JNST — JUMP IF STICKY BIT IS CLEAR**

**Operation:** If the sticky bit flag is clear (i.e., 0), the distance from the end of this instruction to the target label is added to the program counter, effecting the jump. The offset from the end of this instruction to the target label must be in the range of -128 to +127. If the sticky bit flag is set (i.e., 1), control passes to the next sequential instruction.

if  $ST = 0$  then

$PC \leftarrow PC + \text{disp}$  (sign-extended to 16 bits)

**Assembly Language Format:** JNST cadd

**Object Code Format:** [ 11010000 ][ disp ]

Bytes: 2

States: Jump Not Taken: 4

Jump Taken: 8

Flags Affected

Z	N	C	V	VT	ST
-	-	-	-	-	-

**3.13.44. JNV — JUMP IF OVERFLOW FLAG IS CLEAR**

**Operation:** If the overflow flag is clear (i.e., 0), the distance from the end of this instruction to the target label is added to the program counter, effecting the jump. The offset from the end of this instruction to the target label must be in the range of -128 to +127. If the overflow flag is set (i.e., 1), control passes to next sequential instruction.

if V = 0 then

$PC \leftarrow PC + \text{disp}$  (sign-extended to 16 bits)

**Assembly Language Format:** JNV cadd

**Object Code Format:** [ 11010101 ] [ disp ]

Bytes: 2

States: Jump Not Taken: 4

8 Jump Taken: 8

Flags Affected					
Z	N	C	V	VT	ST
-	-	-	-	-	-

**3.13.45. JNVT — JUMP IF OVERFLOW TRAP IS CLEAR**

**Operation:** If the overflow trap flag is clear (i.e., 0), the distance from the end of this instruction to the target label is added to the program counter, effecting the jump. The offset from the end of this instruction to the target label must be in the range of -128 to +127. If the overflow trap flag is set (i.e., 1), control passes to the next sequential instruction.

if VT = 0 then

$PC \leftarrow PC + \text{disp}$  (sign-extended to 16 bits)

**Assembly Language Format:** JNVT cadd

**Object Code Format:** [ 11010100 ] [ disp ]

Bytes: 2

States: Jump Not Taken: 4

8 Jumps Taken: 8

Flags Affected					
Z	N	C	V	VT	ST
-	-	-	-	0	-

**3.13.46. JST — JUMP IF STICKY BIT IS SET**

**Operation:** If the sticky bit flag is set (i.e., 1), the distance from the end of this instruction to the target label is added to the program counter, effecting the jump. The offset from the end of this instruction to the target label must be in the range of -128 to +127. If the sticky bit flag is clear (i.e., 0), control passes to the next sequential instruction.

if ST = 1 then

PC ← PC + disp (sign-extended to 16 bits)

**Assembly Language Format:** JST cadd

**Object Code Format:** [ 11011000 ][ disp ]

Bytes: 2

States: Jump Not Taken: 4

Jump Taken: 8

Flags Affected

Z	N	C	V	VT	ST
-	-	-	-	-	-

**3.13.47. JV — JUMP IF OVERFLOW FLAG IS SET**

**Operation:** If the overflow flag is set (i.e., 1), the distance from the end of this instruction to the target label is added to the program counter, effecting the jump. The offset from the end of this instruction to the target label must be in the range of -128 to +127. If the overflow flag is clear (i.e., 0), control passes to the next sequential instruction.

if V = 1 then

PC ← PC + disp (sign-extended to 16 bits)

**Assembly Language Format:** JV cadd

**Object Code Format:** [ 11011101 ][ disp ]

Bytes: 2

States: Jump Not Taken: 4

Jump Taken: 8

Flags Affected

Z	N	C	V	VT	ST
-	-	-	-	-	-



**3.13.48. JVT — JUMP IF OVERFLOW TRAP IS SET**

**Operation:** If the overflow flag is set (i.e., 1), the distance from the end of this instruction to the target label is added to the program counter, effecting the jump. The offset from the end of this instruction to the target label must be in the range of -128 to +127. If the overflow trap flag is clear (i.e., 0), control passes to the next sequential instruction.

if VT = 1 then

PC ← PC + disp (sign-extended to 16 bits)

**Assembly Language Format:** JVT cadd

**Object Code Format:** [ 11011100 ] [ disp ]

Bytes: 2

States: Jump Not Taken: 4

Jump Taken: 8

Flags Affected					
Z	N	C	V	VT	ST
-	-	-	-	0	-

**3.13.49. LCALL — LONG CALL**

**Operation:** The contents of the program counter (the return address) is pushed onto the stack. Then the distance from the end of this instruction to the target label is added to the program counter, effecting the call. The operand may be any address in the entire address space.

SP ← SP - 2

(SP) ← PC

PC ← PC + disp

**Assembly Language Format:** LCALL cadd

**Object Code Format:** [ 11101111 ] [ disp-low ] [ disp-hi ]

Bytes: 3

States: Onchip stack: 13

Offchip stack: 16

Flags Affected					
Z	N	C	V	VT	ST
-	-	-	-	-	-

**3.13.50. LD — LOAD WORD**

**Operation:** The value of the source (rightmost) word operand is stored into the destination (leftmost) operand.

$(\text{DEST}) \leftarrow (\text{SRC})$

**Assembly Language Format:**      DST      SRC  
LD    wreg,    waop

**Object Code Format:** [ 101000aa ][ waop ][ wreg ]

Bytes: 2 + BEA

States: 4 + CEA

Flags Affected					
Z	N	C	V	VT	ST
-	-	-	-	-	-

**3.13.51. LDB — LOAD BYTE**

**Operation:** The value of the source (rightmost) byte operand is stored into the destination (leftmost) operand.

$(\text{DEST}) \leftarrow (\text{SRC})$

**Assembly Language Format:**      DST      SRC  
LDB   breg,    baop

**Object Code Format:** [ 101100aa ][ baop ][ breg ]

Bytes: 2 + BEA

States: 4 + CEA

Flags Affected					
Z	N	C	V	VT	ST
-	-	-	-	-	-

**3.13.52. LDBSE — LOAD INTEGER WITH SHORT-INTEGER**

**Operation:** The value of the source (rightmost) byte operand is sign-extended and stored into the destination (leftmost) word operand.

(low byte DEST)  $\leftarrow$  (SRC)  
 if (SRC) < 80H then  
   (high byte DEST)  $\leftarrow$  0  
 else  
   (high byte DEST)  $\leftarrow$  0FFH  
 end \_ if

**Assembly Language Format:**           DST    SRC  
 LDBSE   wreg,   baop

**Object Code Format:** [ 101111aa ][ baop ][ wreg ]

Bytes: 2+BEA

States: 4+CEA

Flags Affected					
Z	N	C	V	VT	ST
—	—	—	—	—	—

**3.13.53. LDBZE — LOAD WORD WITH BYTE**

**Operation:** The value of the source (rightmost) byte operand is zero-extended and stored into the destination (leftmost) word operand.

(low byte DEST)  $\leftarrow$  (SRC)  
 (high byte DEST)  $\leftarrow$  0

**Assembly Language Format:**           DST    SRC  
 LDBZE   wreg,   baop

**Object Code Format:** [ 101011aa ][ baop ][ wreg ]

Bytes: 2+BEA

States: 4+CEA

Flags Affected					
Z	N	C	V	VT	ST
—	—	—	—	—	—

**3.13.54. LJMP — LONG JUMP**

**Operation:** The distance from the end of this instruction to the target label is added to the program counter, effecting the jump. The operand may be any address in the entire address space.

$$PC \leftarrow PC + \text{disp}$$

**Assembly Language Format:** LJMP cadd

**Object Code Format:** [ 11100111 ][ disp-low ][ disp-hi ]

Bytes: 3

States: 8

Flags Affected					
Z	N	C	V	VT	ST
-	-	-	-	-	-

**3.13.55. MUL (Two Operands) — MULTIPLY INTEGERS**

**Operation:** The two INTEGER operands are multiplied using signed arithmetic and the 32-bit result is stored into the destination (leftmost) LONG-INTEGER operand. The sticky bit flag is undefined after the instruction is executed.

$$(\text{DEST}) \leftarrow (\text{DEST}) * (\text{SRC})$$

**Assembly Language Format:**

	DST	SRC
MUL	lreg,	waop

**Object Code Format:** [ 11111110 ][ 011011aa ][ waop ][ lreg ]

Bytes: 3 + BEA

States: 29 + CEA

Flags Affected					
Z	N	C	V	VT	ST
-	-	-	-	-	?



**3.13.56. MUL (Three Operands) — MULTIPLY INTEGERS**

**Operation:** The second and third INTEGER operands are multiplied using signed arithmetic and the 32-bit result is stored into the destination (leftmost) INTEGER operand. The sticky bit flag is undefined after the instruction is executed.

$$(DEST) \leftarrow (SRC1) * (SRC2)$$

**Assembly Language Format:**

DST    SRC1   SRC2  
MUL   lreg,   wreg,   waop

**Object Code Format:** [ 11111110 ][ 010011aa ][ waop ][ wreg ][ lreg ]

Bytes: 4 + BEA

States: 30 + CEA

Flags Affected					
Z	N	C	V	VT	ST
—	—	—	—	—	?

**3.13.57. MULB (Two Operands) — MULTIPLY SHORT-INTEGERS**

**Operation:** The two SHORT-INTEGERS operands are multiplied using signed arithmetic and the 16-bit result is stored into the destination (leftmost) INTEGER operand. The sticky bit flag is undefined after the instruction is executed.

$$(DEST) \leftarrow (DEST) * (SRC)$$

**Assembly Language Format:**

DST    SRC  
MULB   wreg,   baop

**Object Code Format:** [ 11111110 ][ 011111aa ][ baop ][ wreg ]

Bytes: 3 + BEA

States: 21 + CEA

Flags Affected					
Z	N	C	V	VT	ST
—	—	—	—	—	?

**3.13.58. MULB (Three Operands) — MULTIPLY SHORT-INTEGERS**

**Operation:** The second and third SHORT-INTEGER operands are multiplied using signed arithmetic and the 16-bit result is stored into the destination (leftmost) INTEGER operand. The sticky bit flag is undefined after the instruction is executed.

$$(DEST) \leftarrow (SRC1) * (SRC2)$$

**Assembly Language Format:** `DST SRC1 SRC2`  
`MULB wreg, breg baop`

**Object Code Format:** [ 11111110 ] [ 010111aa ] [ baop ] [ breg ] [ wreg ]

Bytes: 4 + BEA  
 States: 22 + CEA

Flags Affected					
Z	N	C	V	VT	ST
—	—	—	—	—	?

**3.13.59. MULU (Two Operands) — MULTIPLY WORDS**

**Operation:** The two WORD operands are multiplied using unsigned arithmetic and the 32-bit result is stored into the destination (leftmost) DOUBLE-WORD operand. The sticky bit flag is undefined after the instruction is executed.

$$(DEST) \leftarrow (DEST) * (SRC)$$

**Assembly Language Format:** `DST SRC`  
`MULU lreg, waop`

**Object Code Format:** [ 011011aa ] [ waop ] [ lreg ]

Bytes: 2 + BEA  
 States: 25 + CEA

Flags Affected					
Z	N	C	V	VT	ST
—	—	—	—	—	?

**3.13.60. MULU (Three Operands) — MULTIPLY WORDS**

**Operation:** The second and third WORD operands are multiplied using unsigned arithmetic and the 32-bit result is stored into the destination (leftmost) DOUBLE-WORD operand. The sticky bit flag is undefined after the instruction is executed.

$$(\text{DEST}) \leftarrow (\text{SRC1}) * (\text{SRC2})$$

**Assembly Language Format:** DST SRC1 SRC2  
MULU lreg, wreg, waop

**Object Code Format:** [ 010011aa ][ waop ][ wreg ][ lreg ]

Bytes: 3 + BEA  
States: 26 + CEA

Flags Affected					
Z	N	C	V	VT	ST
-	-	-	-	-	?

**3.13.61. MULUB (Two Operands) — MULTIPLY BYTES**

**Operation:** The two BYTE operands are multiplied using unsigned arithmetic and the WORD result is stored into the destination (leftmost) operand. The sticky bit flag is undefined after the instruction is executed.

$$(\text{DEST}) \leftarrow (\text{DEST}) * (\text{SRC})$$

**Assembly Language Format:** DST SRC  
MULUB wreg, baop

**Object Code Format:** [ 011111aa ][ baop ][ wreg ]

Bytes: 2 + BEA  
States: 17 + CEA

Flags Affected					
Z	N	C	V	VT	ST
-	-	-	-	-	?

**3.13.62. MULUB (Three Operands) — MULTIPLY BYTES**

**Operation:** The second and third BYTE operands are multiplied using unsigned arithmetic and the WORD result is stored into the destination (leftmost) operand. The sticky bit flag is undefined after the instruction is executed.

$$(\text{DEST}) \leftarrow (\text{SRC1}) * (\text{SRC2})$$

**Assembly Language Format:**

```
DST SRC1 SRC2
MULUB wreg, breg, baop
```

**Object Code Format:** [ 010111aa ][ baop ][ breg ][ wreg ]

Bytes: 3 + BEA

States: 18 + CEA

Flags Affected					
Z	N	C	V	VT	ST
-	-	-	-	-	?

**3.13.63. NEG — NEGATE INTEGER**

**Operation:** The value of the INTEGER operand is negated.

$$(\text{DEST}) \leftarrow -(\text{DEST})$$

**Assembly Language Format:** NEG wreg

**Object Code Format:** [ 00000011 ][ wreg ]

Bytes: 2

States: 4

Flags Affected					
Z	N	C	V	VT	ST
✓	✓	✓	✓	↑	-



### 3.13.64. NEGB — NEGATE SHORT-INTEGER

**Operation:** The value of the SHORT-INTEGER operand is negated.  
 $(\text{DEST}) \leftarrow -(\text{DEST})$

**Assembly Language Format:** NEGB breg

**Object Code Format:** [ 00010011 ][ breg ]

Bytes: 2

States: 4

Flags Affected					
Z	N	C	V	VT	ST
✓	✓	✓	✓	↑	—

### 3.13.65. NOP — NO OPERATION

**Operation:** Nothing is done. Control passes to the next sequential instruction.

**Assembly Language Format:** NOP

**Object Code Format:** [ 11111101 ]

Bytes: 1

States: 4

Flags Affected					
Z	N	C	V	VT	ST
—	—	—	—	—	—

## 3.13.66. NORML — NORMALIZE LONG-INTEGERS

**Operation:** The LONG-INTEGERS operand is normalized; i.e., it is shifted to the left until its most significant bit is 1. If the most significant bit is still 0 after 31 shifts, the process stops and the zero flag is set. The number of shifts actually performed is stored in the second operand.

```
(COUNT) ← 0
do while (MSB(DEST) = 0) AND ((COUNT) < 31)
    (DEST) ← (DEST) * 2
    (COUNT) ← (COUNT) + 1
end _ while
```

**Assembly Language Format:** NORML lreg,breg

**Object Code Format:** [ 00001111 ][ breg ][ lreg ]

Bytes: 3

States: 8 + No. of shifts performed

Flags Affected					
Z	N	C	V	VT	ST
✓	1	0	—	—	—

## 3.13.67. NOT — COMPLEMENT WORD

**Operation:** The value of the WORD operand is complemented: each 1 is replaced with a 0, and each 0 with a 1.

```
(DEST) ← NOT(DEST)
```

**Assembly Language Format:** NOT wreg

**Object Code Format:** [ 00000010 ][ wreg ]

Bytes: 2

States: 4

Flags Affected					
Z	N	C	V	VT	ST
✓	✓	0	0	—	—

**3.13.68. NOTB — COMPLEMENT BYTE**

**Operation:** The value of the BYTE operand is complemented: each 1 is replaced with a 0, and each 0 with a 1.

(DEST) ← NOT (DEST)

**Assembly Language Format:** NOTB breg

**Object Code Format:** [ 00010010 ][ breg ]

Bytes: 2

States: 4

Flags Affected					
Z	N	C	V	VT	ST
✓	✓	0	0	—	—

**3.13.69. OR — LOGICAL OR WORDS**

**Operation:** The source (rightmost) WORD is ORed with the destination (leftmost) WORD operand. Each bit is set to 1 if the corresponding bit in either the source operand or the destination operand is 1. The result replaces the original destination operand.

(DEST) ← (DEST) OR (SRC)

**Assembly Language Format:** DST SRC  
OR wreg, waop

**Object Code Format:** [ 100000aa ][ waop ][ wreg ]

Bytes: 2 + BEA

States: 4 + CEA

Flags Affected					
Z	N	C	V	VT	ST
✓	✓	0	0	—	—

### 3.13.70. ORB — LOGICAL OR BYTES

**Operation:** The source (rightmost) BYTE operand is ORed with the destination (leftmost) BYTE operand. Each bit is set to 1 if the corresponding bit in either the source operand or the destination operand was 1. The result replaces the original destination operand.

(DEST) ← (DEST) OR (SRC)

**Assembly Language Format:** ORB breg,baop

**Object Code Format:** [ 100100aa ][ baop ][ breg ]

Bytes: 2 + BEA

States: 4 + CEA

Flags Affected					
Z	N	C	V	VT	ST
✓	✓	0	0	-	-

### 3.13.71. POP — POP WORD

**Operation:** The word on top of the stack is popped and placed at the destination operand.

(DEST) ← (SP)

SP ← SP + 2

**Assembly Language Format:** POP waop

**Object Code Format:** [ 110011aa ][ waop ]

Bytes: 1 + BEA

States: Onchip Stack: 12 + CEA

Offchip Stack: 14 + CEA

Flags Affected					
Z	N	C	V	VT	ST
-	-	-	-	-	-



**3.13.72. POPF — POP FLAGS**

**Operation:** The word on top of the stack is popped and placed in the PSW. Interrupt calls cannot occur immediately following this instruction.

(PSW) ← (SP)  
SP ← SP + 2

**Assembly Language Format:** POPF (OR (DEST) → (DEST))

**Object Code Format:** [ 11110011 ]

Bytes: 1

States: Onchip Stack: 9

Offchip Stack: 13

Flags Affected

Z	N	C	V	VT	ST
✓	✓	✓	✓	✓	✓

**3.13.73. PUSH — PUSH WORD**

**Operation:** The specified operand is pushed onto the stack.

SP ← SP - 2

(SP) ← (DEST)

**Assembly Language Format:** PUSH waop

**Object Code Format:** [ 110010aa ] [ waop ]

Bytes: 1 + BEA

States: Onchip Stack: 8 + CEA

Offchip Stack: 12 + CEA

Flags Affected

Z	N	C	V	VT	ST
-	-	-	-	-	-

**3.13.74. PUSHF — PUSH FLAGS**

**Operation:** The PSW is pushed on top of the stack, and then set to all zeroes. This implies that all interrupts are disabled. Interrupt-calls cannot occur immediately following this instruction.

$$SP \leftarrow SP - 2$$

$$(SP) \leftarrow PSW$$

$$PSW \leftarrow 0$$

**Assembly Language Format:** PUSHF

**Object Code Format:** [ 11110010 ]

Bytes: 1

States: Onchip Stack: 8

Offchip Stack: 12

Flags Affected					
Z	N	C	V	VT	ST
0	0	0	0	0	0

**3.13.75. RET — RETURN FROM SUBROUTINE**

**Operation:** The PC is popped off the top of the stack.

$$PC \leftarrow (SP)$$

$$SP \leftarrow SP + 2$$

**Assembly Language Format:** RET

**Object Code Format:** [ 11110000 ]

Bytes: 1

States: Onchip Stack: 12

Offchip Stack: 16

Flags Affected					
Z	N	C	V	VT	ST
—	—	—	—	—	—

### 3.13.76. RST — RESET SYSTEM

**Operation:** The PSW is initialized to zero, and the PC is initialized to 2080H. The I/O registers are set to their initial value (see section 2.15.2, "Reset Status"). Executing this instruction will cause a pulse to appear on the reset pin of the 8096.

PSW  $\leftarrow$  0  
PC  $\leftarrow$  2080H

**Assembly Language Format:** RST

**Object Code Format:** [ 11111111 ]

Bytes: 1

States: 16

Flags Affected					
Z	N	C	V	VT	ST
0	0	0	0	0	0

### 3.13.77. SCALL — SHORT CALL

**Operation:** The contents of the program counter (the return address) is pushed onto the stack. Then the distance from the end of this instruction to the target label is added to the program counter, effecting the call. The offset from the end of this instruction to the target label must be in the range of -1024 to +1023 inclusive.

SP  $\leftarrow$  SP - 2  
(SP)  $\leftarrow$  PC  
PC  $\leftarrow$  PC + disp (sign-extended to 16 bits)

**Assembly Language Format:** SCALL cadd

**Object Code Format:** [ 00101xxx ] [ disp-low ]

where xxx holds the three high-order bits of displacement.

Bytes: 2

States: Onchip Stack: 13

Offchip Stack: 16

Flags Affected					
Z	N	C	V	VT	ST
-	-	-	-	-	-

## 3.13.78. SETC — SET CARRY FLAG

**Operation:** The carry flag is set.  
 $C \leftarrow 1$

**Assembly Language Format:** SETC

**Object Code Format:** [ 11111001 ]

Bytes: 1

States: 4

Flags Affected

Z	N	C	V	VT	ST
-	-	1	-	-	-

## 3.13.79. SHL — SHIFT WORD LEFT

**Operation:** The destination (leftmost) word operand is shifted left as many times as specified by the count (rightmost) operand. The count may be specified either as an immediate value in the range of 0 to 15 inclusive, or as the content of any register, the address of which is 16 to 255. The right bits of the result are filled with zeroes. The last bit shifted out is saved in the carry flag.

```
Temp ← (COUNT)
do while Temp >> 0
  C ← High order bit of (DEST)
  (DEST) ← (DEST)*2
  Temp ← Temp — 1
end _ while
```

**Assembly Language Format:** SHL wreg,#count  
 or  
 SHL wreg,breg

**Object Code Format:** [ 00001001 ][ cnt/breg ][ wreg ]

Bytes: 3

States: 7 + No. of shifts performed

note: 0 place shifts take 8 states.

Flags Affected

Z	N	C	V	VT	ST
✓	?	✓	✓	↑	-



**3.13.80. SHLB — SHIFT BYTE LEFT**

**Operation:** The destination (leftmost) byte operand is shifted left as many times as specified by the count (rightmost) operand. The count may be specified either as an immediate value in the range of 0 to 15 inclusive, or as the content of any register, the address of which is 16 to 255. The right bits of the result are filled with zeroes. The last bit shifted out is saved in the carry flag.

```
Temp ← (COUNT)
do while Temp <> 0
  C ← High order bit of (DEST)
  (DEST) ← (DEST)*2
  Temp ← Temp - 1
end _ while
```

**Assembly Language Format:** SHLB breg,#count  
or  
SHLB breg,breg

**Object Code Format:** [ 00011001 ][ cnt/breg ][ breg ]

Bytes: 3

States: 7 + No. of shifts performed

note: 0 place shifts take 8 states.

Flags Affected					
Z	N	C	V	VT	ST
✓	?	✓	✓	↑	-

**Operation:** The destination (leftmost) double-word operand is shifted left as many times as specified by the count (rightmost) operand. The count may be specified either as an immediate value in the range of 0 to 15 inclusive, or as the content of any register, the address of which is 16 to 255. The right bits of the result are filled with zeroes. The last bit shifted out is saved in the carry flag.

```
Temp ← (COUNT)
do while Temp <> 0
    C ← High order bit of (DEST)
    (DEST) ← (DEST)*2
    Temp ← Temp - 1
end _ while
```

**Assembly Language Format:** SHLL lreg,#count  
or  
SHLL lreg,breg

**Object Code Format:** [ 00001101 ] [ cnt/breg ] [ : lreg ]

Bytes: 3

States: 7 + No. of shifts performed

note: 0 place shifts take 8 states.

Flags Affected					
Z	N	C	V	VT	ST
✓	?	✓	✓	↑	-

Flags Affected					
Z	N	C	V	VT	ST
✓	0	✓	0	-	✓

**3.13.82. SHR — LOGICAL RIGHT SHIFT WORD**

**Operation:** The destination (leftmost) word operand is shifted right as many times as specified by the count (rightmost) operand. The count may be specified either as an immediate value in the range of 0 to 15 inclusive, or as the content of any register, the address of which is 16 to 255. The left bits of the result are filled with zeroes. The last bit shifted out is saved to the carry. The sticky bit flag is cleared at the beginning of the instruction, and set if at any time during the shift a 1 is shifted first into the carry flag, and a further shift cycle occurs.

```
Temp ← (COUNT)
do while Temp <> 0
  C ← Low order bit of (DEST)
  (DEST) ← (DEST) / 2 where / is unsigned division
  Temp ← Temp - 1
end _ while
```

**Assembly Language Format:** SHR wreg,#count

or

SHR wreg,breg

**Object Code Format:** [ 00001000 ][ cnt/breg ][ wreg ]

Bytes: 3

States: 7 + No. of shifts performed

note: 0 place shifts take 8 states.

Flags Affected

Z	N	C	V	VT	ST
✓	0	✓	0	-	✓

**3.13.83. SHRA — ARITHMETIC RIGHT SHIFT WORD**

**Operation:** The destination (leftmost) word operand is shifted right as many times as specified by the count (rightmost) operand. The count may be specified either as an immediate value in the range of 0 to 15 inclusive, or as the content of any register, the address of which is 16 to 255. If the original high order bit value was 0, zeroes are shifted in. If the value was 1, ones are shifted in. The last bit shifted out is saved in the carry. The sticky bit flag is cleared at the beginning of the instruction, and set if at any time during the shift a 1 is shifted first into the carry flag, and a further shift cycle occurs.

```
Temp ← (COUNT)
do while Temp <> 0
  C ← Low order bit of (DEST)
  (DEST) ← (DEST) / 2 where / is signed division
  Temp ← Temp - 1
end _ while
```

**Assembly Language Format:** SHRA wreg,#count  
or  
SHRA wreg,breg

**Object Code Format:** [ 00001010 ][ cnt/breg ][ wreg ]

Bytes: 3

States: 7 + No. of shifts performed

note: 0 place shifts take 8 states.

Flags Affected					
Z	N	C	V	VT	ST
✓	✓	✓	0	-	✓



**3.13.84. SHRAB — ARITHMETIC RIGHT SHIFT BYTE**

**Operation:** The destination (leftmost) byte operand is shifted right as many times as specified by the count (rightmost) operand. The count may be specified either as an immediate value in the range of 0 to 15 inclusive, or as the content of any register, the address of which is 16 to 255. If the original high order bit value was 0, zeroes are shifted in. If that value was 1, ones are shifted in. The last bit shifted out is saved in the carry. The sticky bit flag is cleared at the beginning of the instruction, and set if at any time during the shift a 1 is shifted first into the carry flag, and a further shift cycle occurs.

```
Temp ← (COUNT)
do while Temp <> 0
  C, = Low order bit of (DEST)
  (DEST) ← (DEST) / 2 where / is signed division
  Temp ← Temp - 1
end _ while
```

**Assembly Language Format:** SHRAB breg,#count  
or  
SHRAB breg,breg

**Object Code Format:** [ 00011010 ][ cnt/breg ][ breg ]

Bytes: 3

States: 7 + No. of shifts performed

note: 0 place shifts take 8 states.

Flags Affected					
Z	N	C	V	VT	ST
✓	✓	✓	0	-	✓

**3.13.85. SHRAL — ARITHMETIC RIGHT SHIFT DOUBLE-WORD**

**Operation:** The destination (leftmost) double-word operand is shifted right as many times as specified by the count (rightmost) operand. The count may be specified either as an immediate value in the range of 0 to 15 inclusive, or as the content of any register, the address of which is 16 to 255. If the original high order bit value was 0, zeroes are shifted in. If the value was 1, ones are shifted in. The sticky bit is cleared at the beginning of the instruction, and set if at any time during the shift a 1 is shifted first into the carry flag, and a further shift cycle occurs.

```
Temp ← (COUNT)
do while Temp < > 0
  C ← Low order bit of (DEST)
  (DEST) ← (DEST) / 2 where / is signed division
  Temp ← Temp - 1
end _ while
```

**Assembly Language Format:** SHRAL Ireg,#count  
or  
SHRAL Ireg,breg

**Object Code Format:** [ 00001110 ] [ cnt/breg ] [ Ireg ]

Bytes: 3

States: 7 + No. of shifts performed

note: 0 place shifts take 8 states.

Flags Affected					
Z	N	C	V	VT	ST
✓	✓	✓	0	-	✓

**3.13.86. SHRB — LOGICAL RIGHT SHIFT BYTE**

**Operation:** The destination (leftmost) byte operand is shifted right as many times as specified by the count (rightmost) operand. The count may be specified either as an immediate value in the range of 0 to 15 inclusive, or as the content of any register, the address of which is 16 to 255. The left bits of the result are filled with zeroes. The last bit shifted out is saved in the carry. The sticky bit flag is cleared at the beginning of the instruction, and set if at any time during the shift a 1 is shifted first into the carry flag, and a further shift cycle occurs.

```
Temp ← (COUNT)
do while Temp < > 0
  C ← Low order bit of (DEST)
  (DEST) ← (DEST) / 2 where / is unsigned division
  Temp ← Temp - 1
end _ while
```

**Assembly Language Format:** SHRB breg,#count  
or  
SHRB breg,breg

**Object Code Format:** [ 00011000 ][ cnt/breg ][ breg ]

Bytes: 3

States: 7 + No. of shifts performed

note: 0 place shifts take 8 states.

Flags Affected					
Z	N	C	V	VT	ST
✓	0	✓	0	-	✓

**3.13.87. SHRL — LOGICAL RIGHT SHIFT DOUBLE-WORD**

**Operation:** The destination (leftmost) double-word operand is shifted right as many times as specified by the count (rightmost) operand. The count may be specified either as an immediate value in the range of 0 to 15 inclusive, or as the content of any register, the address of which is 16 to 255. The left bits of the result are filled with zeroes. The last bit shifted out is saved in the carry. The sticky bit flag is cleared at the beginning of the instruction, and set if at any time during the shift a 1 is shifted first into the carry flag, and a further shift cycle occurs.

```
Temp ← (COUNT)
do while Temp < > 0
  C ← Low order bit of (DEST)
  (DEST) ← (DEST) / 2 where/is unsigned division
  Temp ← Temp - 1
end _ while
```

**Assembly Language Format:**

SHRL Ireg, #count

or

SHRL Ireg, breg

**Object Code Format:** [ 00001100 ][ cnt/breg ][ Ireg ]

Bytes: 3

States: 7 + No. of shifts performed

note: 0 place shifts take 8 states.

**Flags Affected**

Z	N	C	V	VT	ST
✓	0	✓	0	—	✓



### 3.13.88. SJMP — SHORT JUMP

**Operation:** The distance from the end of this instruction to the target label is added to the program counter, effecting the jump. The offset from the end of this instruction to the target label must be in the range of -1024 to +1023 inclusive.

$PC \leftarrow PC + \text{disp}$  (sign-extended to 16 bits)

**Assembly Language Format:** SJMP caddr

**Object Code Format:** [ 00100xxx ][ disp-low ]

where xxx holds the three high order bits of the displacement.

Bytes: 2

States: 8

Flags Affected					
Z	N	C	V	VT	ST
-	-	-	-	-	-

### 3.13.89. SKIP — TWO BYTE NO-OPERATION

**Operation:** Nothing is done. This is actually a two-byte NOP where the second byte can be any value, and is simply ignored. Control passes to the next sequential instruction.

**Assembly Language Format:** SKIP breg

**Object Code Format:** [ 00000000 ][ breg ]

Bytes: 2

States: 4

Flags Affected					
Z	N	C	V	VT	ST
-	-	-	-	-	-

**Operation:** The value of the leftmost word operand is stored into the rightmost operand.

$(\text{DEST}) \leftarrow (\text{SRC})$

**Assembly Language Format:**

	SRC	DST
ST	wreg,	waop

**Object Code Format:** [ 110000aa ][ waop ][ wreg ]

Bytes: 2+BEA

States: 4+CEA

Flags Affected					
Z	N	C	V	VT	ST
-	-	-	-	-	-

### 3.13.91. STB — STORE BYTE

**Operation:** The value of the leftmost byte operand is stored into the rightmost operand.

$(\text{DEST}) \leftarrow (\text{SRC})$

**Assembly Language Format:**

	SRC	DST
STB	breg,	baop

**Object Code Format:** [ 110001aa ][ baop ][ breg ]

Bytes: 2+BEA

States: 4+CEA

Flags Affected					
Z	N	C	V	VT	ST
-	-	-	-	-	-

**3.13.92. SUB (Two Operands) — SUBTRACT WORDS**

**Operation:** The source (rightmost) word operand is subtracted from the destination (leftmost) word operand, and the result is stored in the destination. The carry flag is set as complement of borrow.

$$(DEST) \leftarrow (DEST) - (SRC)$$

**Assembly Language Format:**

DST SRC  
SUB wreg, waop

**Object Code Format:** [ 011010aa ][ waop ][ wreg ]

Bytes: 2 + BEA

States: 4 + CEA

Flags Affected					
Z	N	C	V	VT	ST
✓	✓	✓	✓	↑	-

**3.13.93. SUB (Three Operands) — SUBTRACT WORDS**

**Operation:** The source (rightmost) word operand is subtracted from the second word operand, and the result is stored in the destination (the leftmost operand). The carry flag is set as complement of borrow.

$$(DEST) \leftarrow (SRC1) - (SRC2)$$

**Assembly Language Format:**

DST SRC1 SRC2,  
SUB wreg, wreg, waop

**Object Code Format:** [ 010010aa ][ waop ][ Swreg ][ Dwreg ]

Bytes: 3 + BEA

States: 5 + CEA

Flags Affected					
Z	N	C	V	VT	ST
✓	✓	✓	✓	↑	-

**3.13.94. SUBB (Two Operands) — SUBTRACT BYTES**

**Operation:** The source (rightmost) byte is subtracted from the destination (leftmost) byte operand, and the result is stored in the destination. The carry flag is set as complement of borrow.

$$(DEST) \leftarrow (DEST) - (SRC)$$

**Assembly Language Format:**     DST     SRC  
SUBB breg, baop

**Object Code Format:** [ 011110aa ][ baop ][ breg ]

Bytes: 2+BEA

States: 4+CEA

Flags Affected					
Z	N	C	V	VT	ST
✓	✓	✓	✓	↑	-

**3.13.95. SUBB (Three Operands) — SUBTRACT BYTES**

**Operation:** The source (rightmost) byte operand is subtracted from the destination (leftmost) byte operand, and the result is stored in the destination. The carry flag is set as complement of borrow.

$$(DEST) \leftarrow (SRC1) - (SRC2)$$

**Assembly Language Format:**     DST     SRC1     SRC2  
SUBB breg, Sbreg baop

**Object Code Format:** [ 010110aa ][ baop ][ Sbreg ][ Dbreg ]

Bytes: 3+BEA

States: 5+CEA

Flags Affected					
Z	N	C	V	VT	ST
✓	✓	✓	✓	↑	-



**3.13.96. SUBC — SUBTRACT WORDS WITH BORROW**

**Operation:** The source (rightmost) word operand is subtracted from the destination (leftmost) word operand. If the carry flag was clear, 1 is subtracted from the above result. The result replaces the original destination operand. The carry flag is set as complement of borrow.

$$(DEST) \leftarrow (DEST) - (SRC) - (1-C)$$

**Assembly Language Format:**

DST SRC

SUBC wreg, waop

**Object Code Format:** [ 101010aa ][ waop ][ wreg ]

Bytes: 2+BEA

States: 4+CEA

Flags Affected					
Z	N	C	V	VT	ST
↓	✓	✓	✓	↑	—

**3.13.97. SUBCB — SUBTRACT BYTES WITH BORROW**

**Operation:** The source (rightmost) byte operand is subtracted from the destination (leftmost) byte operand. If the carry flag was clear, 1 is subtracted from the above result. The result replaces the original destination operand. The carry flag is set as complement of borrow.

$$(DEST) \leftarrow (DEST) - (SRC) - (1-C)$$

**Assembly Language Format:**

DST SRC

SUBCB breg, baop

**Object Code Format:** [ 101110aa ][ baop ][ breg ]

Bytes: 2+BEA

States: 4+CEA

Flags Affected					
Z	N	C	V	VT	ST
↓	✓	✓	✓	↑	—

**3.13.98. TRAP — SOFTWARE TRAP**

**Operation:** This instruction causes an interrupt-call which is vectored through location 2010H. The operation of this instruction is not effected by the state of the interrupt enable flag in the PSW (I). Interrupt-calls cannot occur immediately following this instruction. This instruction is intended for use by Intel provided development tools. These tools will not support user-application of this instruction.

$$SP \leftarrow SP - 2$$

$$(SP) \leftarrow PC$$

$$PC \leftarrow (2010H)$$

**Assembly Language Format:** This instruction is not supported by revision 1.0 of the 8096 assembly language.

**Object Code Format:** [ 11110111 ]

Bytes: 1

States: Onchip Stack: 21

Offchip Stack: 24

Flags Affected					
Z	N	C	V	VT	ST
-	-	-	-	-	-

**3.13.99. XOR — LOGICAL EXCLUSIVE-OR WORDS**

**Operation:** The source (rightmost) word operand is XORed with the destination (leftmost) word operand. Each bit is set to 1 if the corresponding bit in either the source operand or the destination operand was 1, but not both. The result replaces the original destination operand.

$$(DEST) \leftarrow (DEST) XOR (SRC)$$

**Assembly Language Format:**

	DST	SRC
XOR	wreg,	waop

**Object Code Format:** [ 100001aa ][ waop ][ wreg ]

Bytes: 2 + BEA

States: 4 + CEA

Flags Affected					
Z	N	C	V	VT	ST
✓	✓	0	0	-	-

**Operation:** The source (rightmost) byte operand is XORed with the destination (leftmost) byte operand. Each bit is set to 1 if the corresponding bit in either the source operand or the destination operand was 1, but not both. The result replaces the original destination operand.

(DEST) ← (DEST) XOR (SRC)

**Assembly Language Format:**

DST SRC  
XORB breg, baop

**Object Code Format:** [ 100101aa ][ baop ][ breg ]

Bytes: 2+BEA

States: 4+CEA

Flags Affected					
Z	N	C	V	VT	ST
✓	✓	0	0	-	-

Flags Affected					
Z	N	C	V	VT	ST
-	-	-	-	-	-

### 3.13.33. XOR — LOGICAL EXCLUSIVE-OR WORDS

**Operation:** The source (rightmost) word operand is XORed with the destination (leftmost) word operand. Each bit is set to 1 if the corresponding bit in either the source operand or the destination operand was 1, but not both. The result replaces the original destination operand.

(DEST) ← (DEST) XOR (SRC)

**Assembly Language Format:**  
DST SRC  
XOR wreg, wreg

**Object Code Format:** [ 100001aa ][ wreg ][ wreg ]

Bytes: 2+BEA  
States: 4+CEA

Flags Affected					
Z	N	C	V	VT	ST
✓	✓	0	0	-	-

---

# MCS<sup>®</sup>-96 Hardware Design Information

---

4





## CHAPTER 4

# MCS®-96 HARDWARE DESIGN INFORMATION

### 4.0. HARDWARE INTERFACING OVERVIEW

This section of the manual is devoted to the hardware engineer. All of the information you need to connect the correct pin to the correct external circuit is provided. Many of the special function pins have different characteristics which are under software control, therefore, it is necessary to define the system completely before the hardware is wired-up.

Frequently within this section a specification for a current, voltage, or time period is referred to; the values provided are to be used as an approximation only. The exact specification can be found in the latest data sheet for the particular part and temperature range that is being used.

### 4.1. REQUIRED HARDWARE CONNECTIONS

Although the 8096 is a single-chip microcontroller, it still requires several external connections to make it work. Power must be applied, a clock source provided, and some form of reset circuitry must be present. We will look at each of these areas of circuitry separately. Figure 4-5 shows the connections that are needed for a single-chip system:

#### 4.1.1. Power Supply Information

Power for 8096 flows through 6 pins; one VCC pin, two VSS pins, one VREF (analog VCC), one ANGND (Analog VSS), and one VPD (V Power Down) pin. All six of these pins must be connected to the 8096 for normal operation. The VCC pin, VREF pin and VPD pin should

be tied to 5 volts. When the analog to digital converter is being used it may be desirable to connect the VREF pin to a separate power supply, or at least a separate power supply line.

The two VSS pins should be connected together with as short a lead as possible to avoid problems due to voltage drops across the wiring. There should be no measurable voltage difference between VSS1 and VSS2. The 2 VSS pins and the ANGND pin should all be nominally at 0 volts. The maximum current drain of the 8096 is around 200mA, with all lines unloaded.

When the analog converter is being used, clean, stable power must be provided to the analog section of the chip to assure highest accuracy. To achieve this, it may be desirable to separate the analog power supply from the digital power supply. The VREF pin supplies 5 volts to the analog circuitry and the ANGND pin is the ground for this section of the chip. More information on the analog power supply is in section 4.3.1.

#### 4.1.2. Other Needed Connections

Several of the pins on the 8096 are used to configure the mode of operation. In normal operation the following pins should be tied directly to the indicated power supply.

PIN	POWER SUPPLY
NMI	VCC
$\overline{\text{TEST}}$	VCC
$\overline{\text{EA}}$	VCC (to allow internal execution) VSS (to force external execution)

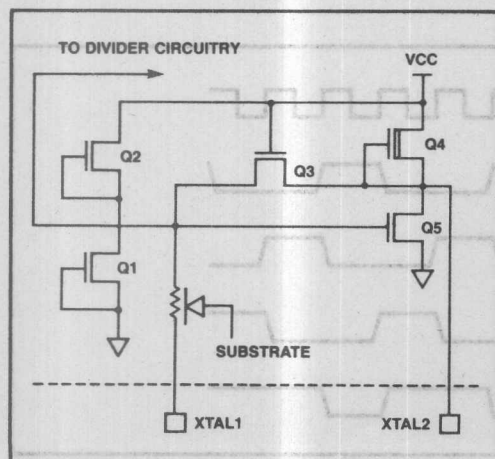


Figure 4-1. 8096 Oscillator Circuit

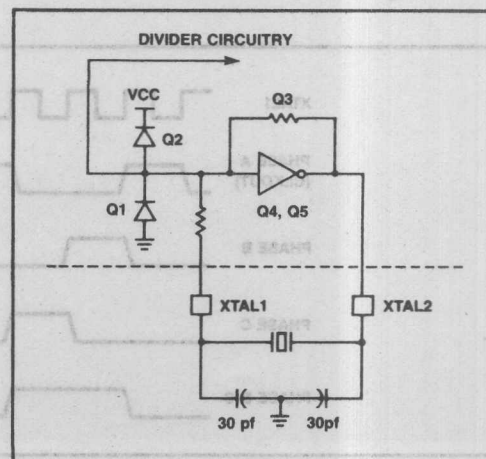


Figure 4-2. Crystal Oscillator Circuit

Although the EA pin has an internal pulldown, it is best to tie this pin to the desired level if it is not left completely disconnected. This will prevent induced noise from disturbing the system.

#### 4.1.3. Oscillator Information

The 8096 requires a clock source to operate. This clock can be provided to the chip through the XTAL1 input or the on-chip oscillator can be used. The frequency of operation is from 6.0 MHz to 12 MHz.

The on-chip circuitry for the 8096 oscillator is a single stage linear inverter as shown in Figure 4-1. It is intended for use as a crystal-controlled, positive reactance oscillator with external connections as shown in Figure 4-2. In this application, the crystal is being operated in its fundamental

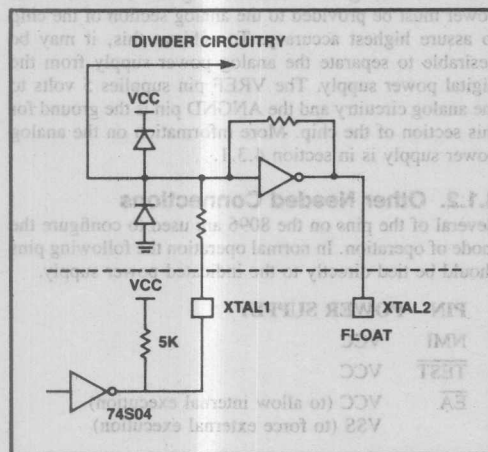


Figure 4-3. External Clock Drive

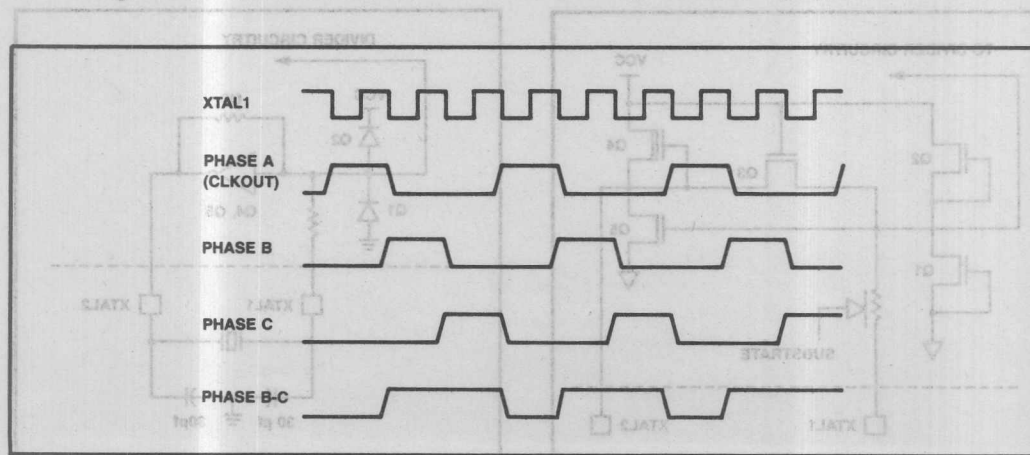


Figure 4-4. Internal Timings

response mode as an inductive reactance in parallel resonance with capacitance external to the crystal.

The crystal specifications and capacitance values (C1 and C2 in Figure 4-2) are not critical. 30 pF can be used in these positions at any frequency with good quality crystals. For 0.5% frequency accuracy, the crystal frequency can be specified at series resonance or for parallel resonance with any load capacitance. (In other words, for that degree of frequency accuracy, the load capacitance simply doesn't matter.) For 0.05% frequency accuracy the crystal frequency should be specified for parallel resonance with 25 pF load capacitance, if C1 and C2 are 30 pF.

A more in-depth discussion of crystal specifications and the selection of values for C1 and C2 can be found in the Intel Application Note, AP-155, "Oscillators for Microcontrollers."

To drive the 8096 with an external clock source, apply the external clock signal to XTAL1 and let XTAL2 float. An example of this circuit is shown in Figure 4-3. The required voltage levels on XTAL1 are specified in the data sheet. The signal on XTAL1 must be clean with good solid levels. It is important that the minimum high and low times are met.

There is no specification on rise and fall times, but they should be reasonably fast (on the order of 30 nanoseconds) to avoid having the XTAL1 pin in the transition range for long periods of time. The longer the signal is in the transition region, the higher the probability that an external noise glitch could be seen by the clock generator circuitry. Noise glitches on the 8096 internal clock lines will cause unreliable operation.

The clock generator provides a 3 phase clock output from the XTAL1 pin input. Figure 4-4 shows the waveforms of the major internal timing signals.

#### 4.1.4. Reset Information

In order for the 8096 to function properly it must be reset. This is done by holding the reset pin low for at least 2 state times after the power supply is within tolerance, the oscillator has stabilized, and the back-bias generator has stabilized. Typically, the back-bias generator requires one millisecond to stabilize.

There are several ways of doing this, the simplest being just to connect a capacitor from the reset pin to ground. The capacitor should be on the order of 1 to 2 microfarads for every millisecond of reset time required. This method will only work if the rise time of VCC is fast and the total reset time is less than around 50 milliseconds. It also may not work if the reset pin is to be used to reset other parts on the board. An 8096 with the minimum required connections is shown in Figure 4-5.

The 8096  $\overline{\text{RESET}}$  pin can be used to allow other chips on the board to make use of the watchdog timer or the RST instruction. When this is done the reset hardware should be a one-shot with an open-collector output. The reset pulse going to the other parts may have to be buffered and lengthened with a one-shot, since the  $\overline{\text{RESET}}$  low duration

is only two state times. If this is done, it is possible that the 8096 will be reset and start running before the other parts on the board are out of reset. The software must account for this possible problem.

A capacitor directly connected to  $\overline{\text{RESET}}$  cannot be used to reset the part if the pin is to be used as an output. If a large capacitor is used, the pin will pull down more slowly than normal. It will continue to pull down until the 8096 is reset. It could fall so slowly that it never goes below the internal switch point of the reset signal (1 to 1.5 volts), a voltage which may be above the guaranteed switch point of external circuitry connected to the pin. Several circuit examples are shown in Figure 4-6.

#### 4.1.5. Sync Mode

If  $\overline{\text{RESET}}$  is brought high at the same time as or just after the rising edge of XTAL1, the part will start executing the 10 state time RST instruction exactly  $6\frac{1}{2}$  XTAL1 cycles later. This feature can be used to synchronize several MCS-96 devices. A diagram of a typical connection is shown in Figure 4-7. It should be noted that parts that start in sync may not stay that way, due to propagation delays which may cause the synchronized parts to receive signals at slightly different times.

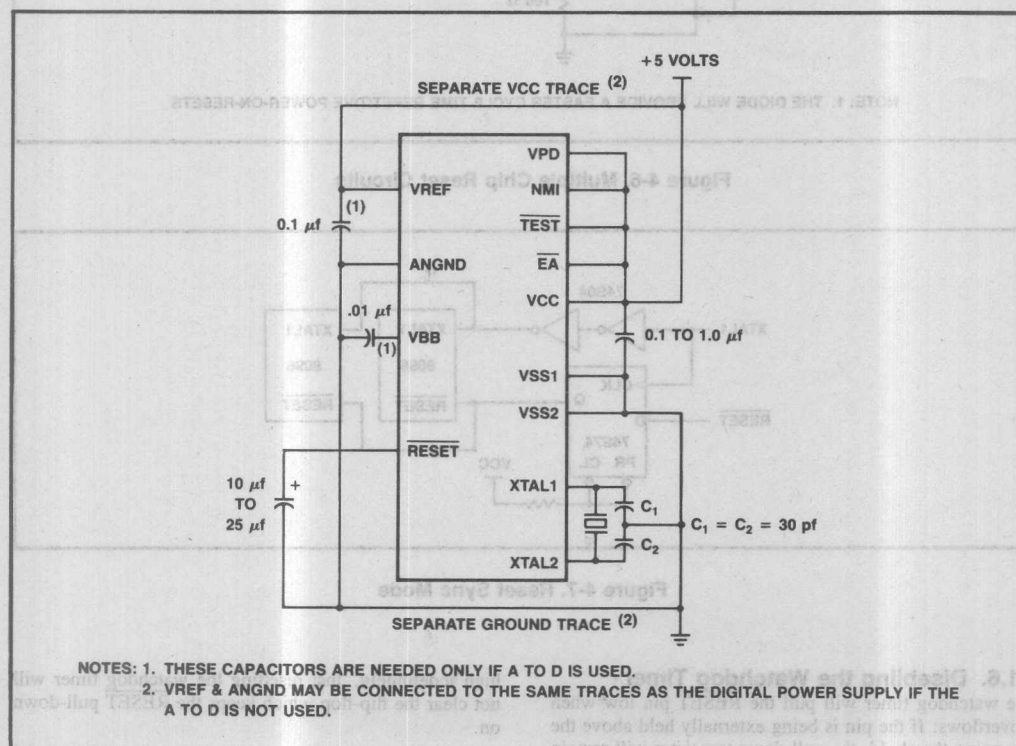


Figure 4-5. Minimum Hardware Connections



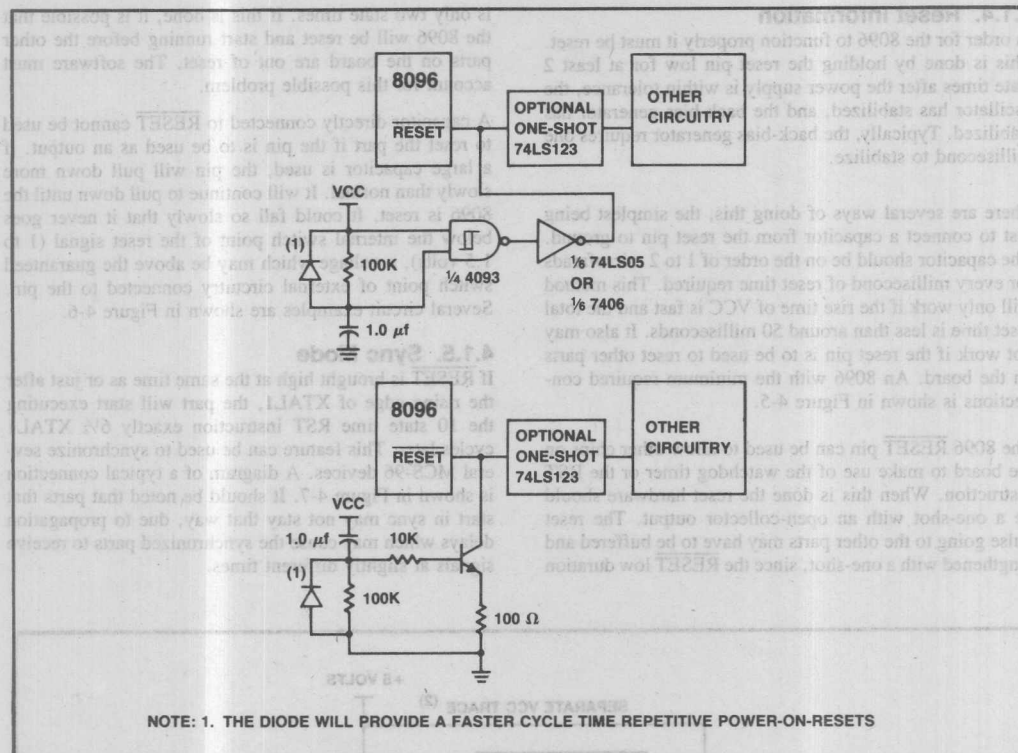


Figure 4-6. Multiple Chip Reset Circuits

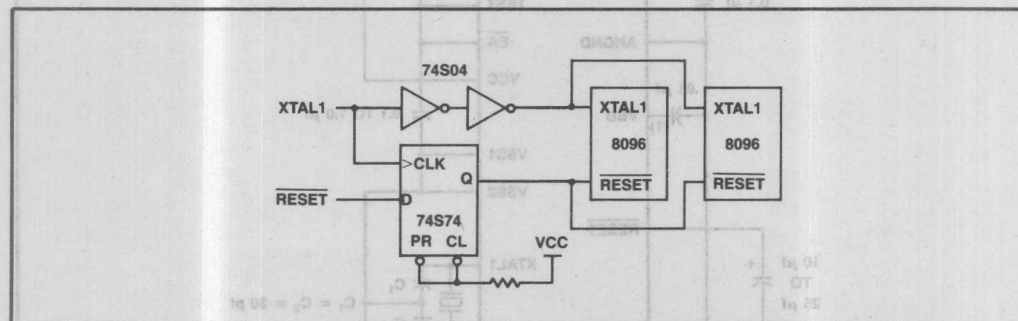


Figure 4-7. Reset Sync Mode

#### 4.1.6. Disabling the Watchdog Timer

The watchdog timer will pull the RESET pin low when it overflows. If the pin is being externally held above the low going threshold, the pull-down transistor will remain on indefinitely. This means that once the watchdog overflows, the part must be reset or RESET must be held

high indefinitely. Just resetting the watchdog timer will not clear the flip-flop which keeps the RESET pull-down on.

The pull-down is capable of sinking on the order of 30 milliamps if it is held at 2.0 volts. This amount of current

may cause some long term reliability problems due to localized chip heating. For this reason, parts that will be used in production should never have had the watchdog timer over-ridden for more than a second or two.

Whenever the reset pin is being pulled high while the pull-down is on, it should be through a resistor that will limit the voltage on RESET to 2.5 volts and the current through the pin to 40 milliamps. Figure 4-8 shows a circuit which will provide the desired results. Using the LED will provide the additional benefit of having a visual indicator that the part is trying to reset itself, although this circuit only works at room temperature and  $V_{CC} = 5$  Volts.

If it is necessary to disable the watchdog timer for more than a brief test the software solution of never initiating the times should be used. See Section 2.14.

#### 4.1.7. Power Down Circuitry

Battery backup can be provided on the 8096 with a 1 mA current drain at 5 volts. This mode will hold locations 0F0H through 0FFH valid as long as the power to the VPD pin remains on. The required timings to put the part into power-down and an overview of this mode are given in section 2.4.2.

A 'key' can be written into power-down RAM while the part is running. This key can be checked on reset to determine if it is a start-up from power-down or a complete cold start. In this way the validity of the power-down RAM can be verified. The length of this key determines the probability that this procedure will work, however, there is always a statistical chance that the RAM will power up with a replica of the key.

Under most circumstances, the power-fail indicator which is used to initiate a power-down condition must come from the unfiltered, unregulated section of the power supply. The power supply must have sufficient storage capacity

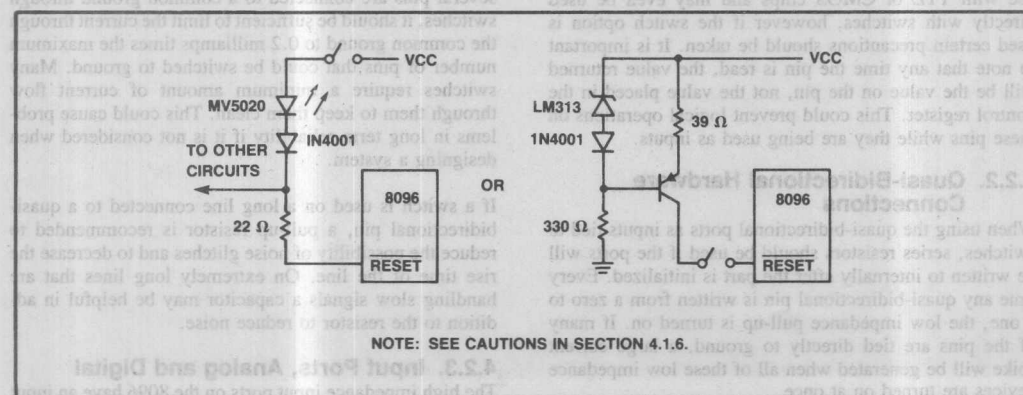


Figure 4-8. Disabling the WDT

to operate the 8096 until it has completed its reset operation.

## 4.2. DRIVE AND INTERFACE LEVELS

There are 5 types of I/O lines on the 8096. Of these, 2 are inputs and 3 are outputs. All of the pins of the same type have the same current/voltage characteristics. Some of the control input pins, such as XTAL1 and RESET, may have slightly different characteristics. These pins are discussed in section 4.1.

While discussing the characteristics of the I/O pins some approximate current or voltage specifications will be given. The exact specifications are available in the latest version of the 8096 Data Sheet.

### 4.2.1. Quasi-Bidirectional Ports

The quasi-bidirectional port is both an input and an output port. It has three states, low impedance current sink, low impedance current source, and high impedance current source. As a low impedance current sink, the pin has a specification of sinking up to around .4 milliamps, while staying below 0.45 volts. The pin is placed in this condition by writing a '0' to the SFR (Special Function Register) controlling the pin.

When a '1' is written to the SFR location controlling the pin, a low impedance current source is turned on for one state time, then it is turned off and the depletion pull-up holds the line at a logical '1' state. The low-impedance pull-up is used to shorten the rise time of the pin, and has current source capability on the order of 100 times that of the depletion pull-up. The configuration of a quasi-bidirectional port pin is shown in Figure 4-9.

While the depletion mode pull-up is the only device on, the pin may be used as an input with a leakage of around

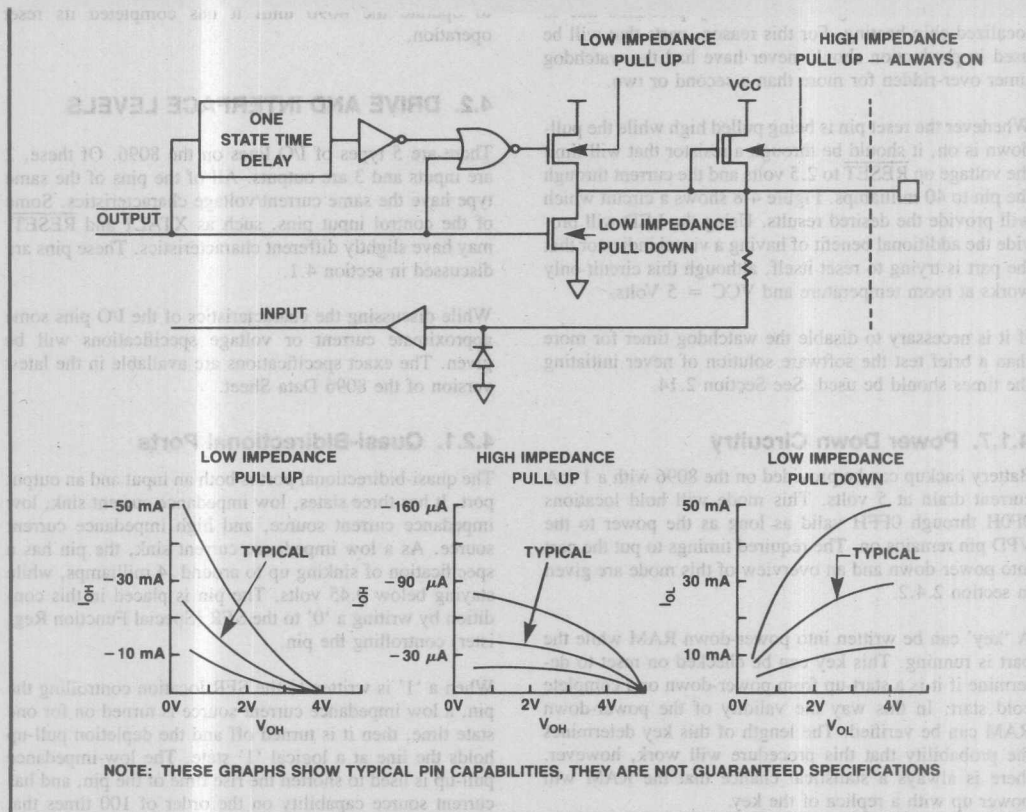


Figure 4-9. Quasi-Bidirectional Port

While the depletion mode pull-up is the only device on the pin, it may be used as an input with a leakage of around 100 microamps from 0.45 volts to VCC. It is ideal for use with TTL or CMOS chips and may even be used directly with switches, however if the switch option is used certain precautions should be taken. It is important to note that any time the pin is read, the value returned will be the value on the pin, not the value placed in the control register. This could prevent logical operations on these pins while they are being used as inputs.

#### 4.2.2. Quasi-Bidirectional Hardware Connections

When using the quasi-bidirectional ports as inputs tied to switches, series resistors should be used if the ports will be written to internally after the part is initialized. Every time any quasi-bidirectional pin is written from a zero to a one, the low impedance pull-up is turned on. If many of the pins are tied directly to ground, a large current spike will be generated when all of these low impedance devices are turned on at once.

For this reason, a series resistor is recommended to limit

the current to a maximum of 0.2 milliamps per pin. If several pins are connected to a common ground through switches, it should be sufficient to limit the current through the common ground to 0.2 milliamps times the maximum number of pins that could be switched to ground. Many switches require a minimum amount of current flow through them to keep them clean. This could cause problems in long term reliability if it is not considered when designing a system.

If a switch is used on a long line connected to a quasi-bidirectional pin, a pull-up resistor is recommended to reduce the possibility of noise glitches and to decrease the rise time of the line. On extremely long lines that are handling slow signals a capacitor may be helpful in addition to the resistor to reduce noise.

#### 4.2.3. Input Ports, Analog and Digital

The high impedance input ports on the 8096 have an input leakage of a few microamps and are predominantly capacitive loads on the order of 10 pf. The Port 0 pins have

an additional function when the A to D converter is being used. These pins are the input to the A to D converter, and as such, are required to provide current to the comparator when a conversion is in process. This means that the input characteristics of a pin will change if a conversion is being done on that pin. See section 4.3.1.

#### 4.2.4. Open Drain Ports

Ports 3 and 4 on the 8096 are open drain ports. There is no pull-up when these pins are used as I/O ports. These pins have different characteristics when used as bus pins as described in the next section. A diagram of the output buffers connected to ports 3 and 4 and the Bus pins is shown in Figure 4-10.

When Ports 3 and 4 are to be used as inputs, or as Bus pins, they must first be written with a '1', this will put the ports in a high impedance mode. When they are used as outputs, a pull-up resistor must be used externally. The sink capability of these pins is on the order of 0.4 milliamps so the total pull-up current to the pin must be less

than this. A 15k pull-up resistor will source a maximum of 0.33 milliamps, so it would be a reasonable value to choose if no other circuits with pullups were connected to the pin.

#### 4.2.5. HSO Pins, Control Outputs and Bus Pins

The control outputs and HSO pins have output buffers with the same output characteristics as those of the bus pins. Included in the category of control outputs are: TXD, RXD (in mode 0), PWM, CLKOUT, ALE, BHE, RD, and WR. The bus pins have 3 states: output high, output low, and high impedance input. As a high output, the pins are specified to source around 200  $\mu$ A to 2.4 volts, but the pins can source on the order of ten times that value in order to provide fast rise times. When used as a low output, the pins can sink around 2 mA at .45 volts, and considerably more as the voltage increases. When in the high impedance state, the pin acts as a capacitive load with a few microamps of leakage. Figure 4-10 shows the internal configuration of a bus pin.

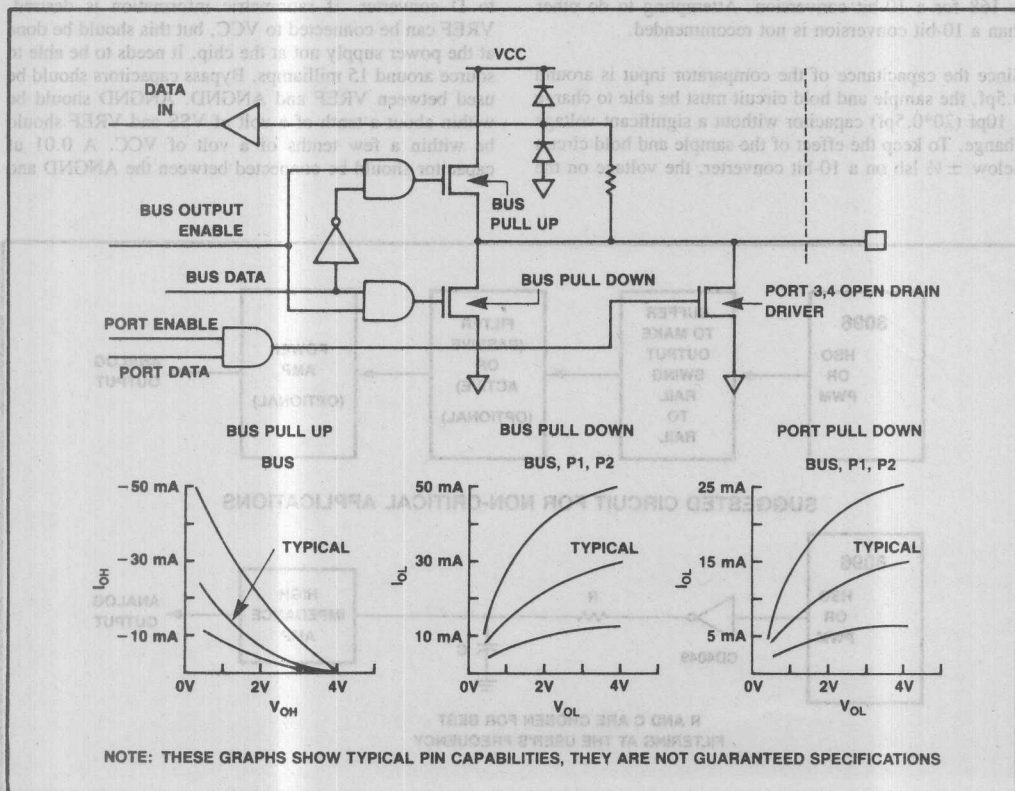


Figure 4-10. Bus and Port 3 and 4 Pins



### 4.3. ANALOG INTERFACE

Interfacing the 8096 to analog signal can be done in several ways. If the 8096 needs to measure an analog signal the A to D converter can be used. Creation of analog outputs can be done with either the PWM output or the HSO unit.

#### 4.3.1. Analog Inputs

The 8096 can have 8 analog inputs and can convert one input at a time into a digital value. Each conversion takes 42 microseconds with a 12 MHz signal on XTAL1. The input signal is applied to one of the Port 0/Analog Channel inputs. Since there is no sample and hold on the A to D, the input signal must remain constant over the sampling period.

When a conversion takes place, the 8096 compares the external signal to that of its internal D to A. Based on the result of the comparison it adjusts the D to A and compares again. Each comparison takes 8 state times and requires the input to the comparator to be charged up. 20 comparisons are made during a conversion, two times for each bit of resolution. An additional 8 states are used to load and store values. The total number of state times required is 168 for a 10-bit conversion. Attempting to do other than a 10-bit conversion is not recommended.

Since the capacitance of the comparator input is around 0.5pf, the sample and hold circuit must be able to charge a 10pf (20\*0.5pf) capacitor without a significant voltage change. To keep the effect of the sample and hold circuit below  $\pm 1/2$  lsb on a 10-bit converter, the voltage on the

sample and hold circuit may vary no more than 0.05% (1/2048).

The effective capacitance of the sample and hold must, therefore, be at least 20000pf or 0.02 uf. If there is external leakage on the capacitor, its value must be increased to compensate for the leakage. At 10μA leakage, 2.5 mV (5/2048) will be lost from a 0.17 uf capacitor in 42 μS. The capacitor connected externally to the pin should, therefore, be at least 0.2 uf for best results. If the external signal changes slowly relative to 42 μS, then a larger capacitor will work well and also filter out unwanted noise.

The converter is a 10-bit, successive approximation, ratiometric converter, so the numerical value obtained from the conversion will be:

$$1023 * (V_{IN} - V_{ANGND}) / (V_{REF} - V_{ANGND})$$

It can be seen that the power supply levels strongly influence the absolute accuracy of the conversion. For this reason, it is recommended that the ANGND pin be tied to a clean ground, as close to the power supply as possible. VREF should be well regulated and used only for the A to D converter. If ratiometric information is desired, VREF can be connected to VCC, but this should be done at the power supply not at the chip. It needs to be able to source around 15 milliamps. Bypass capacitors should be used between VREF and ANGND. ANGND should be within about a tenth of a volt of VSS and VREF should be within a few tenths of a volt of VCC. A 0.01 uf capacitor should be connected between the ANGND and

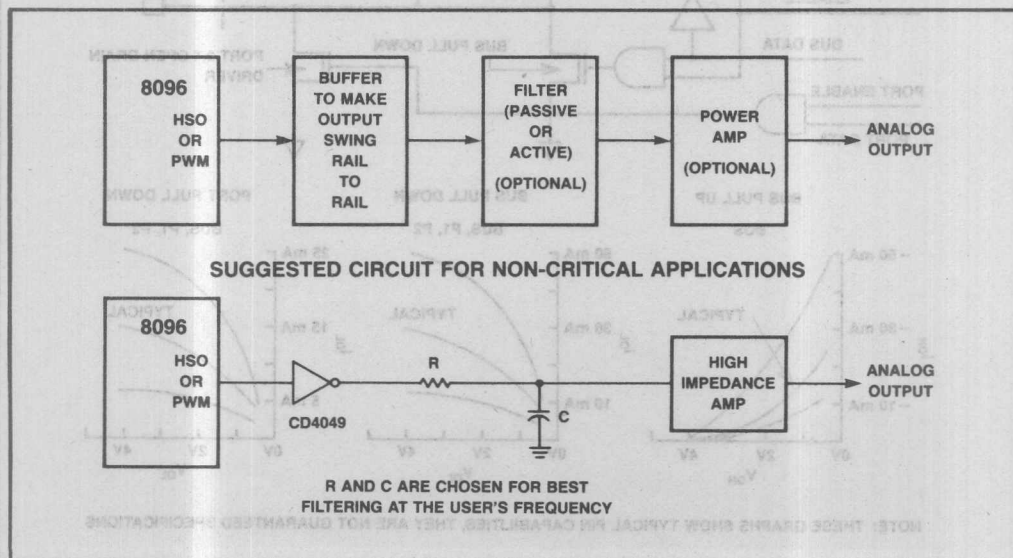


Figure 4-11. D/A Buffer Block Diagram

VBB pins to reduce the noise on VBB and provide the highest possible accuracy. Figure 4-5 shows all of these connections.

### 4.3.2. Analog Output Suggestions

Analog outputs can be generated by two methods, either by using the PWM output or the HSO. Either device will generate a rectangular pulse train that varies in duty cycle and (for the HSO only) period. If a smooth analog signal is desired as an output, the rectangular waveform must be filtered.

In most cases this filtering is best done after the signal is buffered to make it swing from 0 to 5 volts since both of the outputs are guaranteed only to TTL levels. A block diagram of the type of circuit needed is shown in Figure 4-11. By proper selection of components, accounting for temperature and power supply drift, a highly accurate 8-bit D to A converter can be made using either the HSO or the PWM output. If the HSO is used the accuracy could be theoretically extended to 16-bits, however the temperature and noise related problems would be extremely hard to handle.

When driving some circuits it may be desirable to use unfiltered Pulse Width Modulation. This is particularly true for motor drive circuits. The PWM output can be used to generate these waveforms if a fixed period on the order of 64  $\mu$ s is acceptable. If this is not the case then the HSO unit can be used. The HSO can generate a variable waveform with a duty cycle variable in up to 65536 steps and a period of up to 131 milliseconds. Both of these outputs produce TTL levels.

## 4.4. I/O TIMINGS

The I/O pins on the 8096 are sampled and changed at specific times within an instruction cycle. The timings shown in this section are idealized; no propagation delay factors have been taken into account. Designing a system that depends on an I/O pin to change within a window of less than 50 nanoseconds using the information in this section is not recommended.

### 4.4.1. HSO Outputs

Changes in the HSO lines are synchronized to Timer 1. All of the external HSO lines due to change at a certain value of a timer will change just prior to the incrementing of Timer 1. This corresponds to an internal change during Phase C, every eight state times. From an external perspective the HSO pin should change around the rising edge of CLKOUT and be stable by its falling edge. Internal events can occur anytime during the 8 state time window.

Timer 2 is synchronized to increment no faster than Timer 1, so there will always be at least one incrementing of Timer 1 while Timer 2 is at a specific value.

### 4.4.2. HSI Input Sampling

The HSI pins are sampled internally once each state time. Any value on these pins must remain stable for at least 1

full state time to guarantee that it is recognized. This restriction applies even if the divide by eight mode is being used. If two events occur on the same pin within the same 8 state time window, only one of the events will be recorded. If the events occur on different pins they will always be recorded, regardless of the time difference. The 8 state time window, (ie. the amount of time during which Timer 1 remains constant), is stable to within about 20 nanoseconds. The window starts roughly around the rising edge of CLKOUT, however this timing is very approximate due to the amount of internal circuitry involved.

### 4.4.3. Standard I/O Port Pins

Port 0 is different from the other digital ports in that it is actually part of the A to D converter. The port is sampled once every 8 state times, the same frequency at which the comparator is charged-up during an A to D conversion. This 8 state times counter is not synchronized with Timer 1. If this port is used the input signal on the pin must be stable 8 state times prior to reading the SFR.

Port 1 and Port 2 have quasi-bidirectional I/O pins. When used as inputs the data on these pins must be stable one state time prior to reading the SFR. This timing is also valid for the input-only pins of Port 2. When used as outputs, the quasi-bidirectional pins will change state shortly after CLKOUT falls. If the change was from '0' to a '1' the low impedance pull-up will remain on for one state time after the change.

Ports 3 and 4 are addressed as off-chip memory-mapped I/O. The port pins will change state shortly after the rising edge of CLKOUT. When these pins are used as Ports 3 and 4 they are open drain, their structure is different when they are used as part of the bus. See Section 2.12.4.

## 4.5. SERIAL PORT TIMINGS

The serial port on the 8096 was designed to be compatible with the 8051 serial port. Since the 8051 uses a divide by 2 clock and the 8096 uses a divide by 3, the serial port on the 8096 had to be provided with its own clock circuit to maximize its compatibility with the 8051 at high baud rates. This means that the serial port itself does not know about state times. There is circuitry which is synchronized to the serial port and to the rest of the 8096 so that information can be passed back and forth.

The baud rate generator is clocked by either XTAL1 or T2CLK, because T2CLK needs to be synchronized to the XTAL1 signal its speed must be limited to  $\frac{1}{16}$  that of XTAL1. The serial port will not function during the time between the consecutive writes to the baud rate register. Section 2.11.4 discusses programming the baud rate generator.

### 4.5.1. Mode 0

Mode 0 is the shift register mode. The TXD pin sends out a clock train, while the RXD pin transmits or receives the data. Figure 4-12 shows the waveforms and timing. Note that the port starts functioning when a '1' is written

to the REN (Receiver Enable) bit in the serial port control register. If REN is already high, clearing the RI flag will start a reception.

In this mode the serial port can be used to expand the I/O capability of the 8096 by simply adding shift registers.

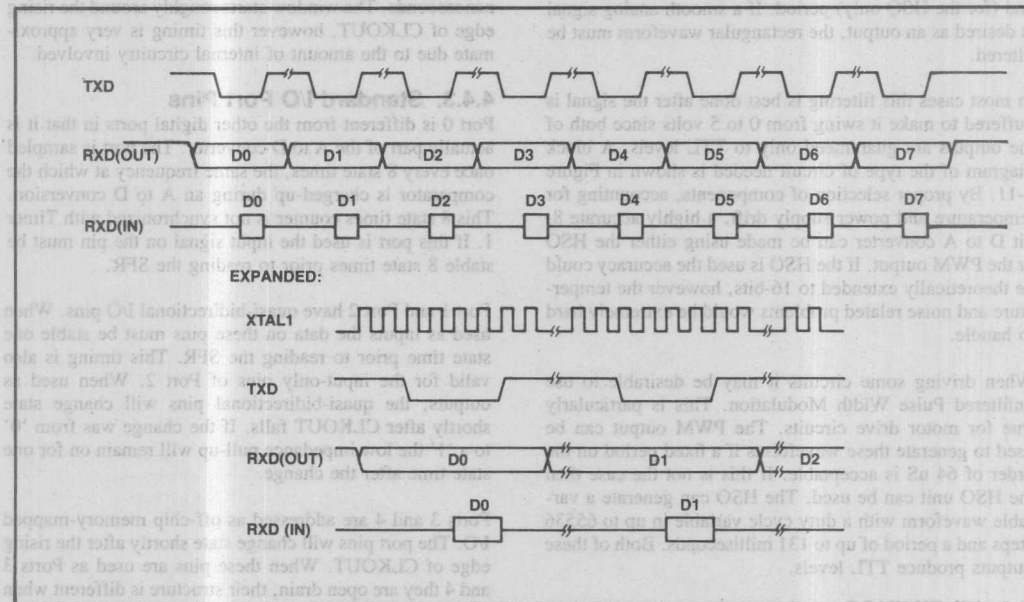


Figure 4-12. Serial Port Timings in Mode 0

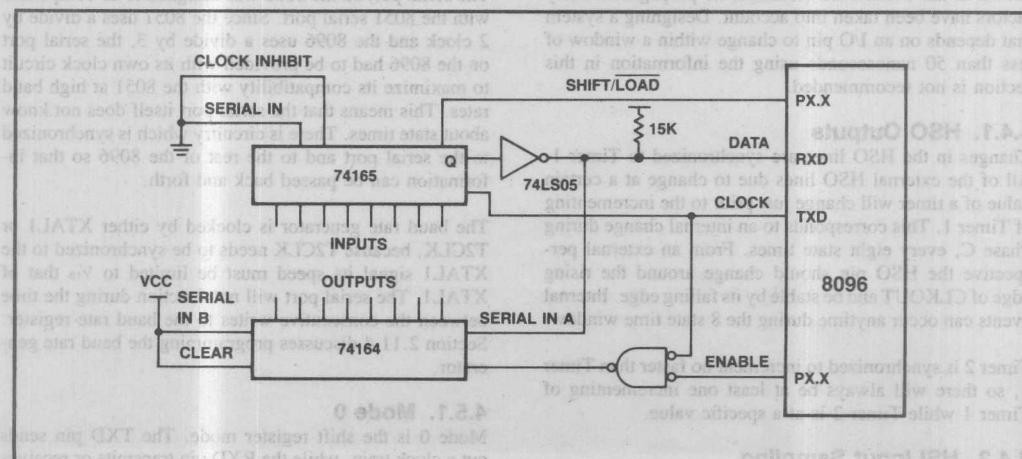
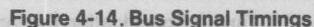


Figure 4-13. Mode 0 Serial Port Example

when a '1 to 0' transition (start bit) is received. The transmit clock may therefore not be in sync with the receive clock, although they will both be at the same frequency.

The TI (Transmit Interrupt) and RI (Receive Interrupt) flags are set to indicate when operations are complete. TI





is set when the last data bit of the message has been sent, not when the stop bit is sent. If an attempt to send another byte is made before the stop bit is sent the port will hold off transmission until the stop bit is complete. RI is set when 8 data bits are received, not when the stop bit is received. Note that when the serial port status register is read both TI and RI are cleared.

Caution should be used when using the serial port to connect more than two devices in half-duplex, (ie. one wire for transmit *and* receive). If the receiving processor does

not wait for one bit time after RI is set before starting to transmit, the stop bit on the link could be squashed. This could cause a problem for other devices listening on the link.

#### 4.5.3. Mode 2 and 3 Timings

Modes 2 and 3 operate in a manner similar to that of mode 1. The only difference is that the data is now made up of 9 bits, so 11-bit packages are transmitted and received. This means that TI and RI will be set on the 9th data bit

Tosc — Oscillator Period, one cycle time on XTAL1.

#### Timings The Memory System Must Meet

TLLYH — ALE low to READY high: Maximum time after ALE falls until READY is brought high to ensure no more wait states. If this time is exceeded unexpected wait states may result. Nominally  $1\text{ Tosc} + 3\text{ Tosc}^*$  number of wait states desired.

TLLYV — ALE low to READY VALID: Maximum time after ALE falls until READY must be valid. If this time is exceeded the part could malfunction necessitating a chip reset. Nominally 2 Tosc periods.

TYLYH — READY low to READY high: Maximum time the part can be in the not-ready state. If it is exceeded, the 8096 dynamic nodes which hold the current instruction may 'forget' how to finish the instruction.

TAVDV — ADDRESS valid to DATA valid: Maximum time that the memory has to output valid data after the 8096 outputs a valid address. Nominally, a maximum of 5 Tosc periods.

TRLDV —  $\overline{\text{READ}}$  low to DATA valid: Maximum time that the memory has to output data after  $\overline{\text{READ}}$  goes low. Nominally, a maximum of 3 Tosc periods.

TRXDZ —  $\overline{\text{READ}}$  not low to DATA float: Time after  $\overline{\text{READ}}$  is no longer low until the memory must float the bus. The memory signal can be removed as soon as  $\overline{\text{READ}}$  is not low, and must be removed within the specified maximum time. Nominally, a maximum of 1 Tosc period.

#### Timings the 8096 Will Provide

TCHCH — CLKOUT high to CLKOUT high: The period of CLKOUT and the duration of one state time. Always 3 Tosc average, but individual periods could vary by a few nanoseconds.

TCHCL — CLKOUT high to CLKOUT low: Nominally 1 Tosc period.

TCLLH — CLKOUT low to ALE high: A help in deriving other timings, typically plus or minus 5 to 10 ns.

TLLCH — ALE low to CLKOUT high: Used to derive other timings, nominally 1 Tosc period.

TLHLL — ALE high to ALE low: ALE pulse width. Useful in determining ALE rising edge to ADDRESS valid time. Nominally 1 Tosc period.

TAVLL — ADDRESS valid to ALE low: Length of time ADDRESS is valid before ALE falls. Important timing for address latch circuitry. Nominally 1 Tosc period.

TLLAX — ALE low to ADDRESS invalid: Length of time ADDRESS is valid after ALE falls. Important timing for address latch circuitry. Nominally 1 Tosc period.

TLLRL — ALE low to  $\overline{\text{READ}}$  or  $\overline{\text{WRITE}}$  low: Length of time after ALE falls before  $\overline{\text{RD}}$  or  $\overline{\text{WR}}$  fall. Could be needed to ensure that proper memory decoding takes place before it is output enabled. Nominally 1 Tosc period.

TRLRH —  $\overline{\text{READ}}$  low to  $\overline{\text{READ}}$  high:  $\overline{\text{RD}}$  pulse width, nominally 1 Tosc period.

TRHLH —  $\overline{\text{READ}}$  high to ALE high: Time between  $\overline{\text{RD}}$  going inactive and next ALE, also used to calculate time between  $\overline{\text{RD}}$  inactive and next ADDRESS valid. Nominally 1 Tosc period.

TWLWH —  $\overline{\text{WRITE}}$  low to  $\overline{\text{WRITE}}$  high: Write pulse width, nominally 2 Tosc periods.

TQVWX — OUTPUT valid to  $\overline{\text{WRITE}}$  not low: time that the OUTPUT data is valid before  $\overline{\text{WR}}$  starts to go high. Nominally 2 Tosc periods.

TWXQX —  $\overline{\text{WRITE}}$  not low to OUTPUT not valid: Time that the OUTPUT data is valid after  $\overline{\text{WR}}$  starts to rise. Nominally 1 Tosc period.

TWXLH —  $\overline{\text{WRITE}}$  not low to ALE high: Time between write starting to rise and next ALE, also used to calculate the time between  $\overline{\text{WR}}$  starting to rise and next ADDRESS valid. Nominally 2 Tosc periods.

Figure 4-15. Timing Specification Explanations

rather than the 8th. The 9th bit can be used for parity or multiple processor communications (see section 2.11).

#### 4.6. BUS TIMING AND MEMORY INTERFACE

##### 4.6.1. Bus Functionality

The 8096 has a multiplexed (address/data) 16 bit bus. There are control lines provided to demultiplex the bus (ALE), indicate reads or writes ( $\overline{RD}$ ,  $\overline{WR}$ ), indicate if the access is for an instruction (INST), and separate the bus into high and low bytes ( $\overline{BHE}$ ,  $\overline{AD0}$ ). Section 2.3.5 contains an overview of the bus operation.

##### 4.6.2. Timing Specifications

Figure 4-14 shows the timing of the bus signals and data lines. Since this is a new part, the exact timing specifications are subject to change, please refer to the latest 8096 data sheet to ensure that your system is designed to the proper specifications. The major timing specifications are described in Figure 4-15.

##### 4.6.3. READY Line Usage

When the processor has to address a memory location that cannot respond within the standard specifications it is necessary to use the READY line to generate wait states. When the READY line is held low the processor waits in a loop for the line to come high. There is a maximum time that the READY line can be held low without risking a processor malfunction due to dynamic nodes that have not been refreshed during the wait states. This time is shown as TYLYH in the data sheet.

In most cases the READY line is brought low after the address is decoded and it is determined that a wait state is needed. It is very likely that some addresses, such as those addressing memory mapped peripherals, would need wait states, and others would not. The READY line must be stable within the TLLYV specification after ALE falls (or the TYVCL before CLKOUT falls) or the processor

could lock-up. There is no requirement as to when READY may go high, as long as the maximum READY low time (TYLYH) is not violated. To ensure that only one wait state is inserted it is necessary to bring READY high TLLYH after the falling edge of ALE.

##### 4.6.4. INST Line Usage

The INST (Instruction) line is high during the output of an address that is for an instruction stream fetch. It is low during the same time for any other memory access. At any other time it is not valid. This pin is not present on the 48-pin versions. The INST signal can be used with a logic analyzer to debug a system. In this way it is possible to determine if the fetch was for instructions or data, making the task of tracing the program much easier.

##### 4.6.5. Address Decoding

The multiplexed bus of the 8096 must be demultiplexed before it can be used. This can be done with 2 74LS373 transparent latches. As explained in section 2.3.5, the latched address signal will be referred to as MA0 through MA15. (Memory Address), and the data lines will be called MD0 through MD15, (Memory Data).

Since the 8096 can make accesses to memory for either bytes or words it is necessary to have a way of determining the type of access desired. The  $\overline{BHE}$  and MA0 lines are used for this purpose.  $\overline{BHE}$  must be latched, as it is valid only when the address is valid. The memory system is typically set up as 32K by 16, instead of 64K by 8. When the  $\overline{BHE}$  line is low, the upper byte is enabled. When MA0 is low, the lower byte is enabled when MA0 is high  $\overline{BHE}$  will be low, and the upper byte is enabled.

When external RAM and EPROM are both used in the system the control logic can be simplified a little to some of the addresses. The 8096 will always output  $\overline{BHE}$  to indicate if a read is of the high byte or the low byte, but it discards the byte it is not going to use. It is therefore possible to use the  $\overline{BHE}$  and MA0 lines only to control

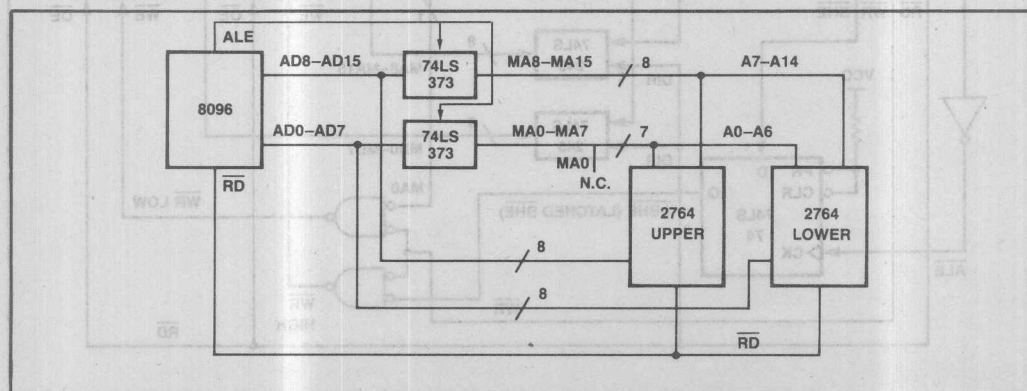


Figure 4-16. Memory Wiring Example—EPROM Only System

signal may be limited by either the ALE timing or the Address timing, these two cases must be considered.

If Limited by ALE

Minimum ALE pulse width =  $\frac{T_{osc}-10}{(TLHL)}$

Minimum Addr set-up to ALE falling = T<sub>osc</sub>-20 (TAVLL)

Total delay from 8096 Address stable to MA (Memory Address) stable would be:

ALE delay from address	- 10
74LS373 clock to output	30
	<hr/>
	20 nanoseconds

If Limited by Address Valid:

In the worst case, the delay in Address valid is controlled by ALE and has a value of 20 nanoseconds.

The address lines are delayed by passing them through the 74LS373s, this delay is specified at 18ns after Address is valid or 30ns after ALE is high. Since the



**Delay of Data Transfer to/from Processor — 12 nanoseconds**

The  $\overline{RD}$  low to Data valid specification (TRLDV) is 3 TOSC-50; (200 ns at 12 MHz). The 74LS245 is enabled by  $\overline{RD}$  and has a delay of 40 ns from enable. The enable delay is clearly not a problem.

The 74LS245 is enabled except during a read, so there is no enable delay to consider for write operations.

The Data In to Data Out delay of the 74LS245 is 12 ns.

**Delay of  $\overline{WR}$  signal to memory — 15 nanoseconds**

Latched  $\overline{BHE}$  is delayed by the inverter on ALE and the 74LS74.

74LS04 delay (Output low to high) = 22

74LS74 delay (Clock to Output) = 40

Delay of Latched  $\overline{BHE}$  from ALE falling = 62 nanoseconds

The 74LS74 requires data valid for 20 ns prior to the clock, the 8096 will have  $\overline{BHE}$  stable TOSC-20 ns (TAVLL, 63 ns at 12 MHz) prior to ALE falling. There is no problem here.

MA0 is valid prior to ALE falling, since the 20 ns Address Delay is less than TAVLL.

$\overline{WR}$  will fall no sooner than TOSC-20 ns (TLLRL, 63 ns at 12 MHz) after ALE goes low. It will therefore be valid just after the Latched  $\overline{BHE}$  is valid, so it is the controlling signal.

$\overline{WR}$  High and  $\overline{WR}$  Low are valid 15 ns after MA0, Latched  $\overline{BHE}$  and  $\overline{WR}$  are valid. Since  $\overline{WR}$  is the last signal to go valid, the delay of  $\overline{WR}$  (High and Low) to memory is 15 ns.

<b>Delay Summary</b> —	Address Delay	= 20 ns
	Data Delay	= 12 ns
	$\overline{WR}$ Delay	= 15 ns
	$\overline{RD}$ Delay	= 0 ns

**Characteristics of a 12 MHz 8096 system with latches:**

Required by system:

Address valid to Data in;

TAVDV	: 386.6 ns maximum
Address Delay	: — 20.0 ns maximum
Data Delay	: — 12.0 ns maximum
	354.6 ns maximum

Read low to Data in;

TRLDV	: 200.0 ns maximum
$\overline{RD}$ Delay	: — 00.0 ns maximum
Data Delay	: — 12.0 ns maximum
	188.0 ns maximum

Provided by System:

Address valid to Control;

TLLRL	: 63.3 ns minimum
TAVLL	: 63.3 ns minimum
Address Delay	: — 20.0 ns maximum
$\overline{WR}$ Delay	: — 00.0 ns minimum (no spec)
	101.6 ns minimum

Write Pulse Width;

TWLWH	: 151.6 ns minimum
Rising $\overline{WR}$ Delay	: — 15.0 ns maximum
Falling $\overline{WR}$ Delay	: — 00.0 ns minimum (no spec)
	146.6 ns minimum

Data Setup to  $\overline{WR}$  rising;

TQVWX	: 136.6 ns minimum
Data Delay	: — 12.0 ns maximum
$\overline{WR}$ Delay	: — 00.0 ns minimum (no spec)
	124.6 ns minimum

Data Hold after  $\overline{WR}$ ;

TWXQX	: 58.3 ns minimum
Data Delay	: — 0.0 ns minimum (no spec)
$\overline{WR}$ Delay	: — 15.0 ns maximum
	43.3 ns minimum

The two memory devices which are expected to be used most often with the 8096 are the 2764 EPROM and the 2128 RAM. The system verification for the 2764 is simple.

**2764 Tac**

(Address valid to Output) < Address valid to Data in  
250 ns < 354 ns O.K.

**2764 Toe**

(Output Enable to Output) < Read low to Data in  
100 ns < 188 ns O.K.

These calculations assume no address decoder delays and no delays on the  $\overline{RD}$  (OE) line. If there are delays in these signals the delays must be added to the 2764's timing.

The read calculations for the 2128 are similar to those for the 2764.

2128-20 Tac < Address valid to Data in  
200 ns < 354 ns O.K.

2128-20 Toe < Read low to Data in  
65 ns < 188 ns O.K.



The write calculations are a little more involved, but still straight-forward.

$$\begin{aligned} 2128 \text{ Twp (Write Pulse)} &< \text{Write Pulse Width} \\ 100 \text{ ns} &< 146 \text{ ns O.K.} \\ 2128 \text{ Tds (Data Setup)} &< \text{Data Setup to } \overline{\text{WR}} \text{ rising} \\ 65 \text{ ns} &< 124 \text{ ns O.K.} \\ 2128 \text{ Tdh (Data Hold)} &< \text{Data Hold after } \overline{\text{WR}} \\ 0 \text{ ns} &< 43 \text{ ns} \end{aligned}$$

All of the above calculations have been done assuming that no components are in the circuit except for those shown in Figure 4-17. If additional components are added, as may be needed for address decoding or memory bank switching, the calculations must be updated to reflect the actual circuit.

#### 4.6.7. I/O Port Reconstruction

When a single-chip system is being designed using a multiple chip system as a prototype, it may be necessary to reconstruct I/O ports 3 and 4 using a memory-mapped I/O technique. The circuit shown in Figure 4-18 provides

this function on the iSBE-96 emulator board. It can be attached to any 8096 system which has the required address decoding and bus demultiplexing.

The output circuitry is basically just a latch that operates when 1FFEh or 1FFFh are placed on the MA lines. The inverters surrounding the latch create an open-collector output to emulate the open-drain output found on the 8096. The 'reset' line is used to set the ports to all 1's when the 8096 is reset. It should be noted that the voltage and current characteristics of this port will differ from those of the 8096, but the basic functionality will be the same.

The input circuitry is just a bus transceiver that is addressed at 1FFEh or 1FFFh. If the ports are going to be used for either input or output, but not both, some of the circuitry can be eliminated.

#### 4.7. NOISE PROTECTION TIPS

Designing controllers differs from designing other computer equipment in the area of noise protection. A microcontroller circuit under the hood of a car, in a photocopier, CRT terminal, or a high speed printer is subject

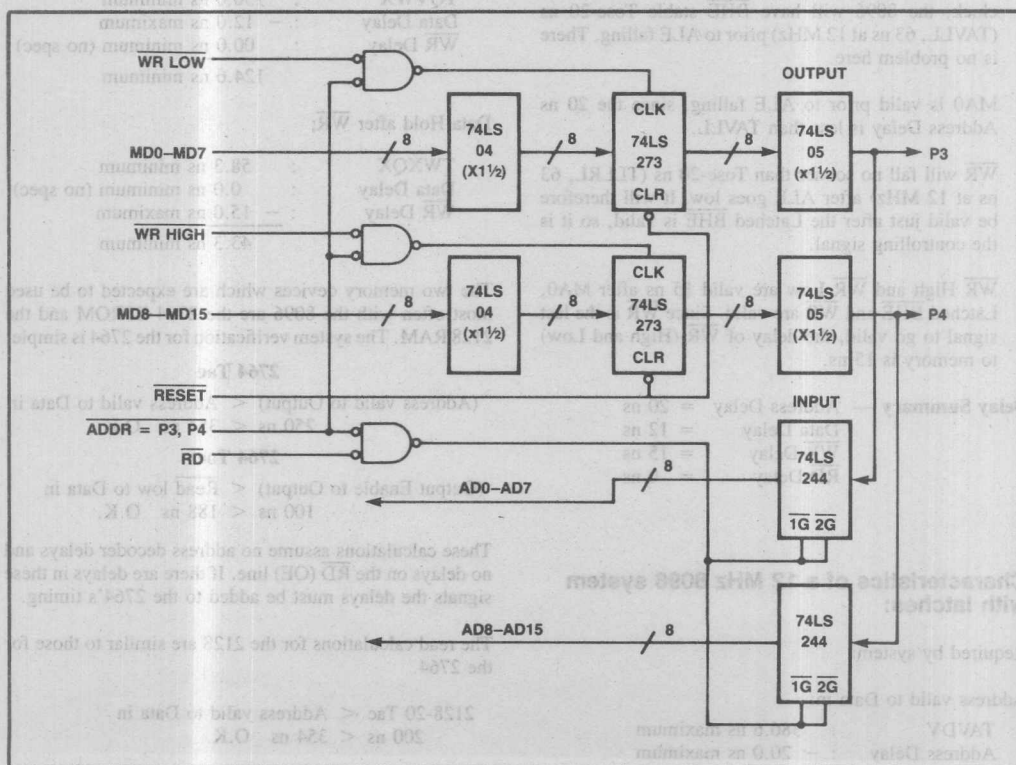


Figure 4-18. I/O Port Reconstruction

to many types of electrical noise. Noise can get to the processor directly through the power supply, or it can be induced onto the board by electromagnetic fields. It is also possible for the pc board to find itself in the path of electrostatic discharges. Glitches and noise on the pc board can cause the processor to act unpredictably, usually by changing either the memory locations or the program counter.

There are both hardware and software solutions to noise problems, but the best solution is good design practice and a few ounces of prevention. The 8096 has a watchdog timer which will reset the part if it fails to execute the software properly. The software should be set up to take advantage of this feature.

It is also recommended that unused areas of code be filled with NOPs and periodic jumps to an error routine or RST (reset chip) instructions. This is particularly important in the code around lookup tables, since if lookup tables are executed all sorts of bad things can happen. Wherever space allows, each table should be surrounded by 7 NOPs (the longest 8096 instruction has 7 bytes) and a RST or jump to error routine instruction. This will help to ensure a speedy recovery should the processor have a glitch in the program flow.

Many hardware solutions exist for keeping pc board noise to a minimum. Ground planes, gridded ground and VCC

structures, bypass capacitors, transient absorbers and power busses with built-in capacitors can all be of great help. It is much easier to design a board with these features than to try to retrofit them later. Proper pc board layout is probably the single most important and, unfortunately, least understood aspect of project design. Minimizing loop areas and inductance, as well as providing clean grounds are very important. More information on protecting against noise can be found in the Intel Application Note AP-125, "Designing Microcontroller Systems For Noisy Environments."

#### 4.8. PACKAGING PINOUTS AND ENVIRONMENT

The MCS-96 family of products is offered in many versions. They are available in 48-pin or 68-pin packages, with or without ROM, and with or without an A to D converter. A summary of the available options is shown in Figure 4-19.

The 48-pin versions are available in a 48-pin DIP (Dual In-Line) package, in either ceramic or plastic.

The 68-pin versions are available in a ceramic pin grid array, and a plastic flatpack. A plastic pin grid array will be available in the near future.

	ROMLESS		WITH ROM	
	68-pin	48-pin	68-pin	48-pin
Without A to D	8096	8094	8396	8394
With A to D	8097	8095	8397	8395

Figure 4-19. The MCS<sup>®</sup>-96 Family of Products

structures, bypass capacitors, transient absorbers and power buses with built-in capacitors can all be of great help. It is much easier to design a board with these features than to try to retrofit them later. Proper pc board layout is probably the single most important and, unfortunately, least understood aspect of project design. Minimizing loop areas and inductance, as well as providing clean grounds are very important. More information on protecting against noise can be found in the Intel Application Note AP-125, "Designing Microcontroller Systems for Noisy Environments."

#### 4.8. PACKAGING PINOUTS AND ENVIRONMENT

The MCS-86 family of products is offered in many versions. They are available in 48-pin or 68-pin packages, with or without ROM, and with or without an A to D converter. A summary of the available options is shown in Figure 4-19.

The 48-pin versions are available in a 48-pin DIP (Dual In-Line) package, in either ceramic or plastic.

The 68-pin versions are available in a ceramic pin grid array, and a plastic package. A plastic pin grid array will be available in the near future.

to many types of electrical noise. Noise can get to the processor directly through the power supply, or it can be induced onto the board by electromagnetic fields. It is also possible for the pc board to find itself in the path of electrostatic discharges. Clutters and noise on the pc board can cause the processor to act unpredictably, usually by changing either the memory location or the program counter.

There are both hardware and software solutions to noise problems, but the best solution is good design practice and a few ounces of prevention. The 8095 has a watchdog timer which will reset the part if it fails to execute the software properly. The software should be set up to take advantage of this feature.

It is also recommended that unused areas of code be filled with NOPs and periodic jumps to an error routine or RST (reset chip) instructions. This is particularly important in the code around lookup tables, since if lookup tables are executed all sorts of bad things can happen. Whatever space allows, each table should be surrounded by 7 NOPs (the longest 8095 instruction has 7 bytes) and a RST or jump to error routine instruction. This will help to ensure a speedy recovery should the processor have a glitch in the program flow.

Many hardware solutions exist for keeping pc board noise to a minimum. Ground planes, gilded ground and VCC

	ROMLESS			WITH ROM	
	68-pin	48-pin	68-pin	68-pin	48-pin
Without A to D	8095	8094	8396	8394	
With A to D	8097	8095	8397	8395	

Figure 4-19. The MCS-86 Family of Products







# 8094/8095/8096/8097

# 8394/8395/8396/8397

## 16-BIT MICROCONTROLLERS

### ■ 839X: an 809X with 8K Bytes of On-chip ROM

- High Speed Pulse I/O
- 10-bit A/D Converter
- 8 Interrupt Sources
- Pulse-Width Modulated Output
- Four 16-bit Software Timers
- 232 Byte Register File
- Memory-to-Memory Architecture
- Full Duplex Serial Port
- Five 8-bit I/O Ports
- Watchdog Timer

The MCS<sup>®</sup>96 family of 16-bit microcontrollers consists of 8 members, all of which are designed for high-speed control functions.

The CPU supports bit, byte, and word operations. 32-bit double-words are supported for a subset of the instruction set. With a 12 MHz input frequency the 8096 can do a 16-bit addition in 1.0  $\mu$ sec and a 16 x 16-bit multiply or 32/16-bit divide in 6.5  $\mu$ sec. Instruction execution times average 1 to 2  $\mu$ sec in typical applications.

Four high-speed trigger inputs are provided to record the times at which external events occur. Six high-speed pulse generator outputs are provided to trigger external events at preset times. The high-speed output unit can simultaneously perform timer functions. Up to four such 16-bit Software Timers can be in operation at once.

An on-chip A/D Converter converts up to 4 (in the 48-pin version) or 8 (in the 68-pin version) analog input channels to 10-bit digital values. This feature is only available on the 8095, 8395, 8097 and 8397.

Also provided on-chip are a serial port, a watchdog timer, and a pulse-width modulated output signal.

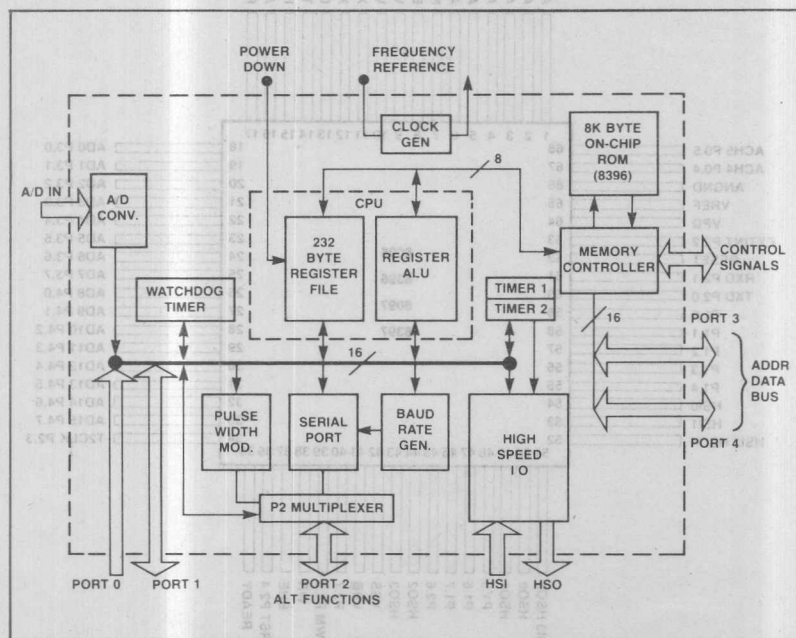


Figure 1. Block Diagram (For simplicity, lines connecting port registers to port buffers are not shown.)

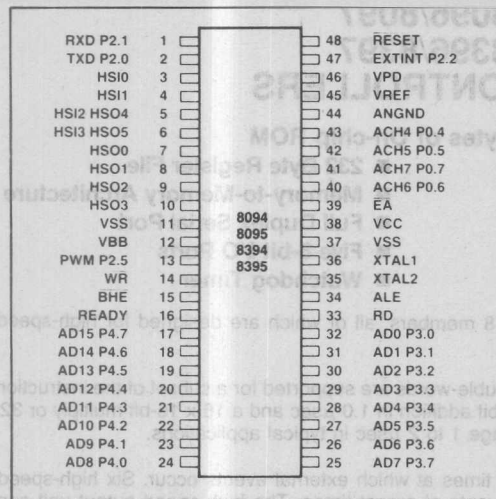


Figure 2. 48-Pin Package

Figure 1 shows a block diagram of the MCS-96 parts, generally referred to as the 8096. The 8096 is available in 48-pin and 68-pin packages, with and without A/D, and with and without on-chip ROM. The MCS-96 numbering system is shown below:

OPTIONS		68 PIN	48 PIN
DIGITAL I/O	ROMLESS	8096	8094
	ROM	8396	8394
ANALOG AND DIGITAL I/O	ROMLESS	8097	8095
	ROM	8397	8395

Figures 2, 3 & 4 show the pinouts for the 48- and 68-pin packages. The 48-pin version is offered in Dual-In-Line package while the 68-pin version comes in a Flat-pack and a Pin Grid Array.

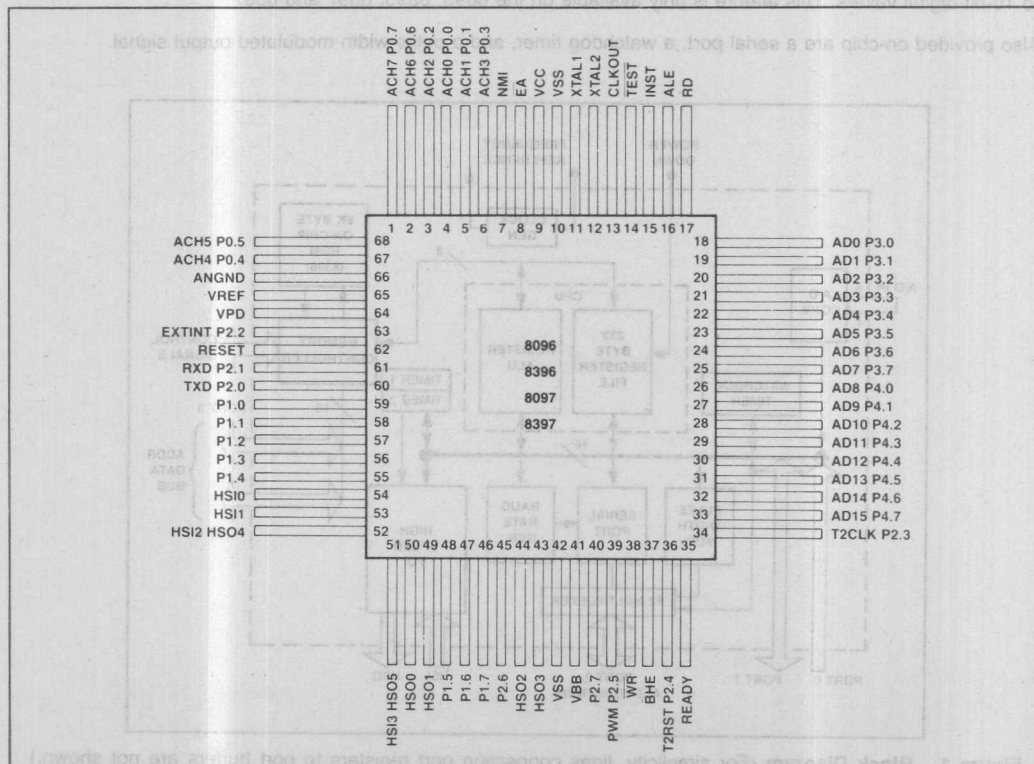


Figure 3. 68-Pin Package (Flat Pack-Top View)

Pins Facing Up											Pins Facing Down												
68	2	4	6	8	10	12	14	16	19	18	17	15	13	11	9	7	5	3	1	68			
66	67								21	20	20	21								67	66		
64	65								23	22	22	23								65	64		
62	63	<b>MCS®-96 68 PIN GRID ARRAY</b>								25	24	24	25	<b>MCS®-96 68 PIN GRID ARRAY</b>								63	62
60	61								27	26	26	27								61	60		
58	59								29	28	28	29								59	58		
56	57								31	30	30	31								57	56		
54	55								33	32	32	33								55	54		
52	53	50	48	46	44	42	40	38	36	34	34	36	38	40	42	44	46	48	50	53	52		
	51	49	47	45	43	41	39	37	35		35	37	39	41	43	45	47	49	51				

**Figure 4. Pin Grid Array**

### Note

1. When the pin grid array package is mounted on the PC board, the pins are numbered counterclockwise as seen from the component side of the board, just like the flatpack when it's mounted in the contactor. Consequently, the PC board layout for pin grid array is compatible with the flat pack in a contactor except for the footprint size. The pin functions (from -1 to -68) on both packages are identical. Refer to Intel's Microcontroller handbook for mechanical dimensions on these packages.

295319



Table 3-1. Instruction Summary

Mnemonic	Operands	Operation (Note 1)	Flags						Notes
			Z	N	C	V	VT	ST	
ADD/ADDB	2	$D \leftarrow D + A$	✓	✓	✓	✓	↑	—	
ADD/ADDB	3	$D \leftarrow B + A$	✓	✓	✓	✓	↑	—	
ADDC/ADDCB	2	$D \leftarrow D + A + C$	↓	✓	✓	✓	↑	—	
SUB/SUBB	2	$D \leftarrow D - A$	✓	✓	✓	✓	↑	—	
SUB/SUBB	3	$D \leftarrow B - A$	✓	✓	✓	✓	↑	—	
SUBC/SUBCB	2	$D \leftarrow D - A + C - 1$	↓	✓	✓	✓	↑	—	
CMP/CMPB	2	$D - A$	✓	✓	✓	✓	↑	—	
MUL/MULU	2	$D, D + 2 \leftarrow D * A$	—	—	—	—	—	?	2
MUL/MULU	3	$D, D + 2 \leftarrow B * A$	—	—	—	—	—	?	2
MULB/MULUB	2	$D, D + 1 \leftarrow D * A$	—	—	—	—	—	?	3
MULB/MULUB	3	$D, D + 1 \leftarrow B * A$	—	—	—	—	—	?	3
DIV/DIVU	2	$D \leftarrow (D, D + 2)/A$ $D + 2$ remainder	—	—	—	✓	↑	—	2
DIVB/DIVUB	2	$D \leftarrow (D, D + 1)/A$ $D + 1$ remainder	—	—	—	✓	↑	—	3
AND/ANDB	2	$D \leftarrow D \text{ and } A$	✓	✓	0	0	—	—	
AND/ANDB	3	$D \leftarrow B \text{ and } A$	✓	✓	0	0	—	—	
OR/ORB	2	$D \leftarrow D \text{ or } A$	✓	✓	0	0	—	—	
XOR/XORB	2	$D \leftarrow D \text{ (excl. or) } A$	✓	✓	0	0	—	—	
LD/LDB	2	$D \leftarrow A$	—	—	—	—	—	—	
ST/STB	2	$A \leftarrow D$	—	—	—	—	—	—	
LDBSE	2	$D \leftarrow A; D + 1 \leftarrow \text{SIGN}(A)$	—	—	—	—	—	—	3,4
LDBZE	2	$D \leftarrow A; D + 1 \leftarrow 0$	—	—	—	—	—	—	3,4
PUSH	1	$SP \leftarrow SP - 2; (SP) \leftarrow A$	—	—	—	—	—	—	
POP	1	$A \leftarrow (SP); SP \leftarrow SP + 2$	—	—	—	—	—	—	
PUSHF	0	$SP \leftarrow SP - 2; (SP) \leftarrow \text{PSW};$ $\text{PSW} \leftarrow 0000\text{H}$ $I \leftarrow 0$	0	0	0	0	0	0	
POPF	0	$\text{PSW} \leftarrow (SP); SP \leftarrow SP + 2$ $I \leftarrow \checkmark$	✓	✓	✓	✓	✓	✓	
SJMP	1	$PC \leftarrow PC + 11\text{-bit offset}$	—	—	—	—	—	—	5
LJMP	1	$PC \leftarrow PC + 16\text{-bit offset}$	—	—	—	—	—	—	5
BR(indirect)	1	$PC \leftarrow (A)$	—	—	—	—	—	—	
SCALL	1	$SP \leftarrow SP - 2; (SP) \leftarrow PC;$ $PC \leftarrow PC + 11\text{-bit offset}$	—	—	—	—	—	—	5
LCALL	1	$SP \leftarrow SP - 2; (SP) \leftarrow PC;$ $PC \leftarrow PC + 16\text{-bit offset}$	—	—	—	—	—	—	5
RET	0	$PC \leftarrow (SP); SP \leftarrow SP + 2$	—	—	—	—	—	—	
J(conditional)	1	$PC \leftarrow PC + 8\text{-bit offset}$	—	—	—	—	—	—	5
JC	1	Jump if $C = 1$	—	—	—	—	—	—	5
JNC	1	Jump if $C = 0$	—	—	—	—	—	—	5

## Note

1. If the mnemonic ends in "B", a byte operation is performed, otherwise a word operation is done. Operands D, B, and A must conform to the alignment rules for the required operand type. D and B are locations in the register file; A can be located anywhere in memory.
2. D, D + 2 are consecutive WORDS in memory; D is DOUBLE-WORD aligned.
3. D, D + 1 are consecutive BYTES in memory; D is WORD aligned.
4. Changes a byte to a word.
5. Offset is a 2's complement number.

Table 3-2. Instruction Summary

Mnemonic	Operands	Operation (Note 1)	Flags						Notes
			Z	N	C	V	VT	ST	
JE	1	Jump if Z = 1	—	—	—	—	—	—	5
JNE	1	Jump if Z = 0	—	—	—	—	—	—	5
JGE	1	Jump if N = 0	—	—	—	—	—	—	5
JLT	1	Jump if N = 1	—	—	—	—	—	—	5
JGT	1	Jump if N = 0 and Z = 0	—	—	—	—	—	—	5
JLE	1	Jump if N = 1 or Z = 1	—	—	—	—	—	—	5
JH	1	Jump if C = 1 and Z = 0	—	—	—	—	—	—	5
JNH	1	Jump if C = 0 or Z = 1	—	—	—	—	—	—	5
JV	1	Jump if V = 1	—	—	—	—	—	—	5
JNV	1	Jump if V = 0	—	—	—	—	—	—	5
JVT	1	Jump if VT = 1; Clear VT	—	—	—	—	0	—	5
JNVT	1	Jump if VT = 0; Clear VT	—	—	—	—	0	—	5
JST	1	Jump if ST = 1	—	—	—	—	—	—	5
JNST	1	Jump if ST = 0	—	—	—	—	—	—	5
JBS	3	Jump if Specified Bit = 1	—	—	—	—	—	—	5,6
JBC	3	Jump if Specified Bit = 0	—	—	—	—	—	—	5,6
DJNZ	1	D ← D - 1; if D ≠ 0 then PC ← PC + 8-bit offset	—	—	—	—	—	—	5
DEC/DECB	1	D ← D - 1	✓	✓	✓	✓	↑	—	
NEG/NEGB	1	D ← 0 - D	✓	✓	✓	✓	↑	—	
INC/INCB	1	D ← D + 1	✓	✓	✓	✓	↑	—	
EXT	1	D ← D; D + 2 ← Sign (D)	✓	✓	0	0	—	—	2
EXTB	1	D ← D; D + 1 ← Sign (D)	✓	✓	0	0	—	—	3
NOT/NOTB	1	D ← Logical Not (D)	✓	✓	0	0	—	—	
CLR/CLRB	1	D ← 0	1	0	0	0	—	—	
SHL/SHLB/SHLL	2	C ← msb — — — — lsb ← 0	✓	?	✓	✓	↑	—	7
SHR/SHRB/SHRL	2	0 → msb — — — — lsb → C	✓	0	✓	0	—	✓	7
SHRA/SHRAB/SHRAL	2	msb → msb — — — — lsb → C	✓	✓	✓	0	—	✓	7
SETC	0	C ← 1	—	—	1	—	—	—	
CLRC	0	C ← 0	—	—	0	—	—	—	
CLRVT	0	VT ← 0	—	—	—	—	0	—	
RST	0	PC ← 2080H	0	0	0	0	0	0	8
DI	0	Disable All Interrupts (I ← 0)	—	—	—	—	—	—	
EI	0	Enable All Interrupts (I ← 1)	—	—	—	—	—	—	
NOP	0	PC ← PC + 1	—	—	—	—	—	—	
SKIP	0	PC ← PC + 2	—	—	—	—	—	—	
NORML	2	Normalize (See sec 3.13.66)	✓	1	0	—	—	—	7
TRAP	0	SP ← SP - 2; (SP) ← PC; PC ← (2010H)	—	—	—	—	—	—	9

## Note

1. If the mnemonic ends in "B", a byte operation is performed, otherwise a word operation is done. Operands D, B and A must conform to the alignment rules for the required operand type. D and B are locations in the register file; A can be located anywhere in memory.
2. Offset is a 2's complement number.
3. Specified bit is one of the 2048 bits in the register file.
4. The "L" (Long) suffix indicates double-word operation.
5. Initiates a Reset by pulling RESET low. Software should re-initialize all the necessary registers with code starting at 2080H.
6. The assembler will not accept this mnemonic.

Table 3-3. Opcode and State Time Listing

MNEMONIC	OPERANDS	DIRECT			IMMEDIATE			INDIRECT®					INDEXED®					
								NORMAL			AUTO-INC.		SHORT			LONG		
		OPCODE	BYTES	STATE TIMES	OPCODE	BYTES	STATE TIMES	OPCODE	BYTES	STATE® TIMES	BYTES	STATE® TIMES	OPCODE	BYTES	STATE® TIMES	BYTES	STATE® TIMES	
ARITHMETIC INSTRUCTIONS																		
ADD	2	64	3	4	65	4	5	66	3	6/11	3	7/12	67	4	6/11	5	7/12	
ADD	3	44	4	5	45	5	6	46	4	7/12	4	8/13	47	5	7/12	6	8/13	
ADDB	2	74	3	4	75	3	4	76	3	6/11	3	7/12	77	4	6/11	5	7/12	
ADDB	3	54	4	5	55	4	5	56	4	7/12	4	8/13	57	5	7/12	6	8/13	
ADDC	2	A4	3	4	A5	4	5	A6	3	6/11	3	7/12	A7	4	6/11	5	7/12	
ADDCB	2	B4	3	4	B5	3	4	B6	3	6/11	3	7/12	B7	4	6/11	5	7/12	
SUB	2	68	3	4	69	4	5	6A	3	6/11	3	7/12	6B	4	6/11	5	7/12	
SUB	3	48	4	5	49	5	6	4A	4	7/12	4	8/13	4B	5	7/12	6	8/13	
SUBB	2	78	3	4	79	3	4	7A	3	6/11	3	7/12	7B	4	6/11	5	7/12	
SUBB	3	58	4	5	59	4	5	5A	4	7/12	4	8/13	5B	5	7/12	6	8/13	
SUBC	2	A8	3	4	A9	4	5	AA	3	6/11	3	7/12	AB	4	6/11	5	7/12	
SUBCB	2	B8	3	4	B9	3	4	BA	3	6/11	3	7/12	BB	4	6/11	5	7/12	
CMP	2	88	3	4	89	4	5	8A	3	6/11	3	7/12	8B	4	6/11	5	7/12	
CMPB	2	98	3	4	99	3	4	9A	3	6/11	3	7/12	9B	4	6/11	5	7/12	
MULU	2	6C	3	25	6D	4	26	6E	3	27/32	3	28/33	6F	4	27/32	5	28/33	
MULU	3	4C	4	26	4D	5	27	4E	4	28/33	4	29/34	4F	5	28/33	6	29/34	
MULUB	2	7C	3	17	7D	3	17	7E	3	19/24	3	20/25	7F	4	19/24	5	20/25	
MULUB	3	5C	4	18	5D	4	18	5E	4	20/25	4	21/26	5F	5	20/25	6	21/26	
MUL	2	②	4	29	②	5	30	②	4	31/36	4	32/37	②	5	31/36	6	32/37	
MUL	3	②	5	30	②	6	31	②	5	32/37	5	33/38	②	6	32/37	7	33/38	
MULB	2	②	4	21	②	4	21	②	4	23/28	4	24/29	②	5	23/28	6	24/29	
MULB	3	②	5	22	②	5	22	②	5	24/29	5	25/30	②	6	24/29	7	25/30	
DIVU	2	8C	3	25	8D	4	26	8E	3	28/32	3	29/33	8F	4	28/32	5	29/33	
DIVUB	2	9C	3	17	9D	3	17	9E	3	20/24	3	21/25	9F	4	20/24	5	21/25	
DIV	2	②	4	29	②	5	30	②	4	32/36	4	33/37	②	5	32/36	6	33/37	
DIVB	2	②	4	21	②	4	21	②	4	24/28	4	25/29	②	5	24/28	6	25/29	

Notes:

- ① Long indexed and Indirect + instructions have identical opcodes with Short indexed and Indirect modes, respectively. The second byte of instructions using any indirect or indexed addressing mode specifies the exact mode used. If the second byte is even, use Indirect or Short Indexed. If it is odd, use Indirect + or Long indexed. In all cases the second byte of the instruction always specifies an even (word) location for the address referenced.
- ② Number of state times shown for internal/external operands.
- ③ The opcodes for signed multiply and divide are the opcodes for the unsigned functions with an "FE" appended as a prefix.

Table 3-3. Continued

MNEMONIC	OPERANDS	DIRECT			IMMEDIATE			INDIRECT®					INDEXED®				
		OPCODE	BYTES	STATE TIMES	OPCODE	BYTES	STATE TIMES	NORMAL			AUTO-INC.		SHORT			LONG	
								OPCODE	BYTES	STATE① TIMES	BYTES	STATE① TIMES	OPCODE	BYTES	STATE① TIMES	BYTES	STATE① TIMES
LOGICAL INSTRUCTIONS																	
AND	2	60	3	4	61	4	5	62	3	6/11	3	7/12	63	4	6/11	5	7/12
AND	3	40	4	5	41	5	6	42	4	7/12	4	8/13	43	5	7/12	6	8/13
ANDB	2	70	3	4	71	3	4	72	3	6/11	3	7/12	73	4	6/11	5	7/12
ANDB	3	50	4	5	51	4	5	52	4	7/12	4	8/13	53	5	7/12	6	8/13
OR	2	80	3	4	81	4	5	82	3	6/11	3	7/12	83	4	6/11	5	7/12
ORB	2	90	3	4	91	3	4	92	3	6/11	3	7/12	93	4	6/11	5	7/12
XOR	2	84	3	4	85	4	5	86	3	6/11	3	7/12	87	4	6/11	5	7/12
XORB	2	94	3	4	95	3	4	96	3	6/11	3	7/12	97	4	6/11	5	7/12
DATA TRANSFER INSTRUCTIONS																	
LD	2	A0	3	4	A1	4	5	A2	3	6/11	3	7/12	A3	4	6/11	5	7/12
LDB	2	B0	3	4	B1	3	4	B2	3	6/11	3	7/12	B3	4	6/11	5	7/12
ST	2	C0	3	4	—	—	—	C2	3	7/11	3	8/12	C3	4	7/11	5	8/12
STB	2	C4	3	4	—	—	—	C6	3	7/11	3	8/12	C7	4	7/11	5	8/12
LDBSE	2	BC	3	4	BD	3	4	BE	3	6/11	3	7/12	BF	4	6/11	5	7/12
LDBZE	2	AC	3	4	AD	3	4	AE	3	6/11	3	7/12	AF	4	6/11	5	7/12
STACK OPERATIONS (internal stack)																	
PUSH	1	C8	2	8	C9	3	8	CA	2	11/15	2	12/16	CB	3	11/15	4	12/16
POP	1	CC	2	12	—	—	—	CE	2	14/18	2	14/18	CF	3	14/18	4	14/18
PUSHF	0	F2	1	8													
POPF	0	F3	1	9													
STACK OPERATIONS (external stack)																	
PUSH	1	C8	2	12	C9	3	12	CA	2	15/19	2	16/20	CB	3	15/19	4	16/20
POP	1	CC	2	14	—	—	—	CE	2	16/20	2	16/20	CF	3	16/20	4	16/20
PUSHF	0	F2	1	12													
POPF	0	F3	1	13													
JUMPS AND CALLS																	
MNEMONIC	OPCODE		BYTES		STATES		MNEMONIC	OPCODE		BYTES		STATES					
LJMP	E7		3		8		LCALL	EF		3		13/16⑤					
SJMP	20-27④		2		8		SCALL	28-2F④		2		13/16⑤					
BR[ ]	E3		2		8		RET	F0		1		12/16⑤					
Notes:							TRAP③	F7		1							

## Notes:

- ① Number of state times shown for internal/external operands.
- ③ The assembler does not accept this mnemonic.
- ④ The least significant 3 bits of the opcode are concatenated with the following 8 bits to form an 11-bit, 2's complement, offset for the relative call or jump.
- ⑤ State times for stack located internal/external.
- ⑥ The assembler uses the generic jump mnemonic (BR) to generate this instruction.



Table 3-4. CONDITIONAL JUMPS

All conditional jumps are 2 byte instructions. They require 8 state times if the jump is taken, 4 if it is not.							
MNEMONIC	OPCODE	MNEMONIC	OPCODE	MNEMONIC	OPCODE	MNEMONIC	OPCODE
JC	DB	JE	DF	JGE	D6	JGT	D2
JNC	D3	JNE	D7	JLT	DE	JLE	DA
JH	D9	JV	DD	JVT	DC	JST	D8
JNH	D1	JNV	D5	JNVT	D4	JNST	D0

## JUMP ON BIT CLEAR OR BIT SET

These instructions are 3-byte instructions. They require 9 state times if the jump is taken, 5 if it is not.									
BIT NUMBER									
MNEMONIC	0	1	2	3	4	5	6	7	8
JBC	30	31	32	33	34	35	36	37	38
JBS	38	39	3A	3B	3C	3D	3E	3F	40

## LOOP CONTROL

DJNZ	OPCODE EO;	3 BYTES;	5/9 STATE TIMES (NOT TAKEN/TAKEN)
------	------------	----------	-----------------------------------

## SINGLE REGISTER INSTRUCTIONS

MNEMONIC	OPCODE	BYTES	STATES	MNEMONIC	OPCODE	BYTES	STATES
DEC	05	2	4	EXT	06	2	4
DECB	15	2	4	EXTB	16	2	4
NEG	03	2	4	NOT	02	2	4
NEGB	13	2	4	NOTB	12	2	4
INC	07	2	4	CLR	01	2	4
INCB	17	2	4	CLRB	11	2	4

## SHIFT INSTRUCTIONS

INSTR		WORD		INSTR		BYTE		INSTR		DBL WD		STATE TIMES
MNEMONIC	OP	B	MNEMONIC	OP	B	MNEMONIC	OP	B	MNEMONIC	OP	B	
SHL	09	3	SHLB	19	3	SHLL	0D	3				7 + 1 PER SHIFT <sup>⑦</sup>
SHR	08	3	SHRB	18	3	SHRL	0C	3				7 + 1 PER SHIFT <sup>⑦</sup>
SHRA	0A	3	SHRAB	1A	3	SHRAL	0E	3				7 + 1 PER SHIFT <sup>⑦</sup>

## SPECIAL CONTROL INSTRUCTIONS

MNEMONIC	OPCODE	BYTES	STATES	MNEMONIC	OPCODE	BYTES	STATES
SETC	F9	1	4	DI	FA	1	4
CLRC	F8	1	4	EI	FB	1	4
CLRVT	FC	1	4	NOP	FD	1	4
RST	FF	1	16	SKIP	00	2	4

## NORMALIZE

NORML	0F	3	11 + 1 PER SHIFT
-------	----	---	------------------

## Notes:

⑥ This instruction takes 2 states to pull RST low, then holds it low for 2 states to initiate a reset. The reset takes 12 states, at which time the program restarts at location 2080H.

⑦ Execution will take at least 8 states, even for 0 shift.

## ELECTRICAL CHARACTERISTICS

## ABSOLUTE MAXIMUM RATINGS

Ambient Temperature Under Bias	-40°C to +70°C
Storage Temperature	-40°C to +150°C
Voltage from Any Pin to VSS or ANGND	-0.3V to +7.0V
Average Output Current from Any Pin	10 mA
Power Dissipation	1.5 Watts

\*NOTICE: Stresses above those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.

## OPERATING CONDITIONS

Symbol	Parameter	Min	Max	Units
TA	Ambient Temperature Under Bias	0	+70	C
VCC	Digital Supply Voltage	4.50	5.50	V
VREF	Analog Supply Voltage	4.5	5.5	V
		VCC - 0.3	VCC + 0.3	V
fOSC	Oscillator Frequency	6.0	12	MHz
VPD	Power-Down Supply Voltage	4.50	5.50	V

VBB should be connected to ANGND through a 0.01  $\mu$ F capacitor. ANGND and VSS should be nominally at the same potential.

## DC CHARACTERISTICS

Symbol	Parameter	Min	Max	Units	Test Conditions
VIL	Input Low Voltage	-0.3	+0.8	V	
VIH	Input High Voltage (Except RESET)	2.0	VCC + 0.5	V	
VIH1	Input High Voltage, RESET Rising	2.4	VCC + 0.5	V	
VIH2	Input High Voltage, RESET Falling	2.0	VCC + 0.5	V	
VOL	Output Low Voltage		0.45	V	See Note 1.
VOH	Output High Voltage	2.4		V	See Note 2.
ICC	VCC Supply Current		200	mA	All outputs disconnected.
IPD	VPD Supply Current		1	mA	Normal operation and Power-Down.
IREF	VREF Supply Current		15	mA	
ILI	Input Leakage Current to all pins of HSI, P0, P3, P4, and to P2.1.		$\pm 10$	$\mu$ A	Vin = 0 to VCC See Note 3
IIH	Input High Current to EA		100	$\mu$ A	VIH = 2.4V
IIL	Input Low Current to all pins of P1, and to P2.6, P2.7.		-100	$\mu$ A	VIL = 0.45V
IIL1	Input Low Current to RESET		-2	mA	VIL = 0.45V
IIL2	Input Low Current P2.2, P2.3, P2.4, READY		-50	$\mu$ A	VIL = 0.45V
Cs	Pin Capacitance (Any Pin to VSS)		10	pF	fTEST = 1MHz

## NOTES:

1. IOL = 0.36 mA for all pins of P1, for P2.6 and P2.7, and for all pins of P3 and P4 when used as ports.  
IOL = 2.0 mA for TXD, RXD (in serial port mode 0), PWM, CLKOUT, ALE, BHE, RD, WR, and all pins of HSO and P3 and P4 when used as external memory bus (AD0-AD15).
2. IOH = -20  $\mu$ A for all pins of P1, for P2.6 and P2.7.  
IOH = -200  $\mu$ A for TXD, RXD (in serial port mode 0), PWM, CLKOUT, ALE, BHE, WR, and all pins of HSO and P3 and P4 when used as external memory bus (AD0-AD15).  
P3 and P4, when used as ports, have open-drain outputs.
3. Analog Conversion not in process.

## A/D CONVERTER SPECIFICATIONS

A/D Converter operation is verified only on the 8097, 8397, 8095, 8395.

The absolute conversion accuracy is dependent on the accuracy of VREF. The specifications given below assume adherence to the Operating Conditions section of these data sheets. Testing is done at VREF = 5.120 volts.

Resolution .....	$\pm 0.001$ VREF
Accuracy .....	$\pm 0.004$ VREF
Differential nonlinearity .....	$\pm 0.002$ VREF max
Integral nonlinearity .....	$\pm 0.004$ VREF max
Channel-to-channel matching .....	$\pm 1$ LSB
Crosstalk (DC to 100kHz) .....	-60dB max

## AC CHARACTERISTICS

Test Conditions: Load capacitance on output pins = 80pf  
Oscillator Frequency = 12.00 MHz  
4.50 Volts, = VCC, = 5.50 Volts; 0°C, = Temperature, = 70°C

Timing Requirements (other system components must meet these specs)

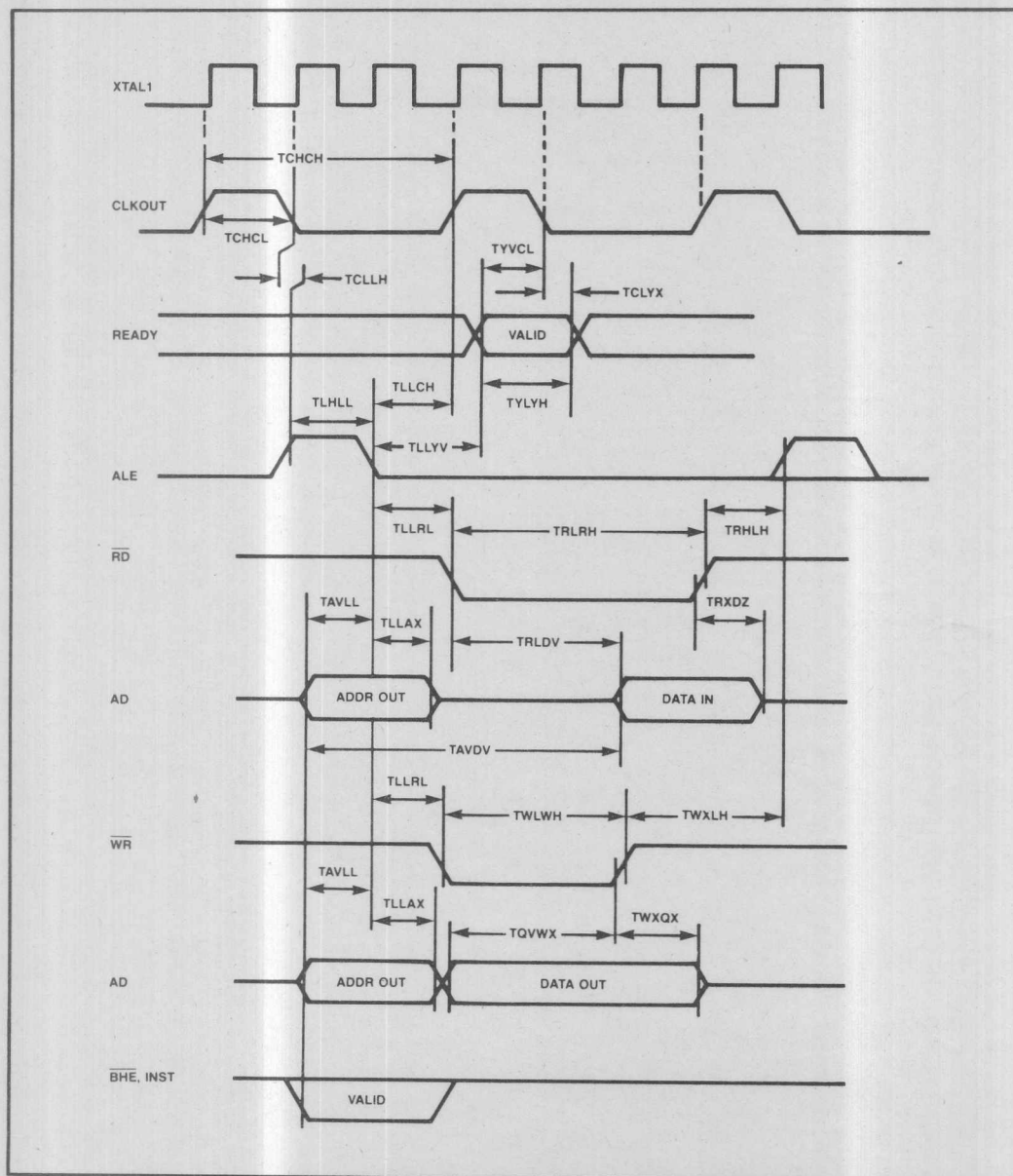
Symbol	Parameter	Min	Max	Units
TCLYX	READY Hold after CLKOUT falling edge	0 (1)		nsec
TLLYV	End of ALE to READY Setup		2Tosc - 60	nsec
TLLYH	End of ALE to READY high		4Tosc - 60 (2)	nsec
TYLYH	Non-ready time		1000	nsec
TAVDV	Address Valid to Input Data Valid		5Tosc - 80	nsec
TRLDV	RD/ Active to Input Data Valid		3Tosc - 60	nsec
TRXDZ	End of RD/ to Input Data Float	0	Tosc - 20	nsec

Timing Responses (MCS®-96 parts meet these specs)

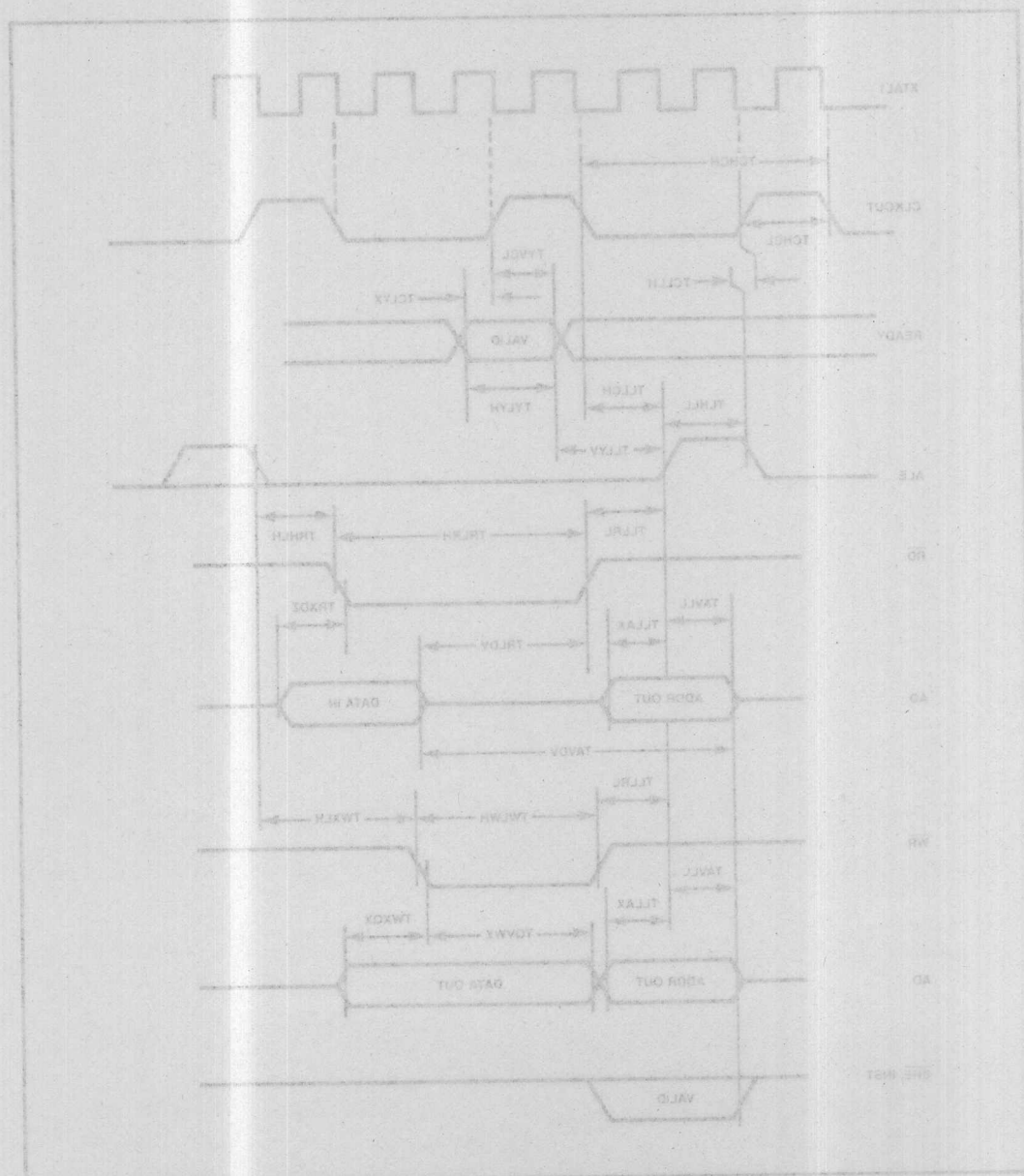
Symbol	Parameter	Min	Max	Units
FXTAL	Oscillator Frequency	6.00	12.00	MHz
Tosc	Oscillator Period	83	166	nsec
TCHCH	CLKOUT Period	3Tosc (3)	3Tosc (3)	nsec
TCHCL	CLKOUT High Time	Tosc - 20	Tosc + 20	nsec
TCLLH	CLKOUT Low to ALE High	-5	20	nsec
TLLCH	ALE Low to CLKOUT High	Tosc - 20	Tosc + 40	nsec
TLHLL	ALE Pulse Width	Tosc - 25	Tosc + 15	nsec
TAVLL	Address Setup to End of ALE	Tosc - 50		nsec
TLLRL	End of ALE to RD/ or WR/ active	Tosc - 20		nsec
TLLAX	Address hold after End of ALE	Tosc - 20		nsec
TWLWH	WR/ Pulse Width	2Tosc - 35		nsec
TQVWX	Output Data Setup to End of WR/	2Tosc - 60		nsec
TWXQX	Output Data Hold after End of WR/	Tosc - 25		nsec
TWXLH	End of WR/ to next ALE	2Tosc - 30		nsec
TRLRH	RD/ Pulse Width	3Tosc - 30		nsec
TRHLH	End of RD/ to next ALE	Tosc - 25		nsec

### NOTES:

1. If the 48-pin part is being used then this timing can be generated by assuming that the CLKOUT falling edge has occurred at  $2Tosc + 55$  (TLLCH(max) + TCHCL(max)) after the falling edge of ALE.
2. If more than one wait state is desired, add 3Tosc for each additional wait state.
3. CLKOUT is directly generated as a divide by 3 of the oscillator. The period will be  $3Tosc \pm 5$  nsec if Tosc is constant and the rise and fall times on XTAL 1 are less than 10 nsec.









6

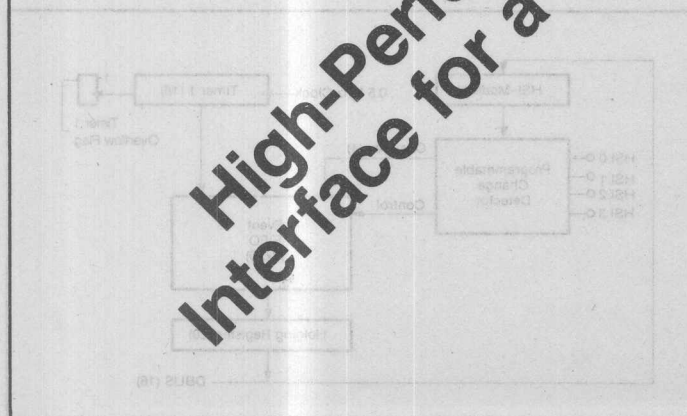
MCS®-96 Article Reprint

February 1984

February 1984

The purpose of the High Speed Input Unit is to allow the measurement of the periods of incoming pulses or frequencies. It is a 16-bit counter that can be programmed to count with high resolution and minimal software overhead. A block diagram of the hardware used to accomplish this goal is shown in Figure 1. The heart of this unit is a programmable logic device which monitors

# rent computer





# High-Performance Event Interface For A Microcomputer

*As silicon technology advances to provide denser geometries, timer structures have become more elegant and powerful.*

Microcontrollers are microprocessors specially configured to monitor and control mechanisms and processes rather than manipulate data. The systems they are imbedded in are often called real time control systems; microcontrollers always incorporate some form of timer structure to allow synchronization with the outside or 'real' world. As silicon technology advances to provide denser geometries, these structures have become more elegant and powerful.

This trend can be seen in the Intel 8048, the Motorola 6801, and the Intel 8051 which were introduced at approximately two and a half year intervals starting in 1976. The 8048 has a single 8-bit timer; the 6801 has a 16-bit timer, and the 8051 has two 16-bit timers. The new 16-bit microcontroller from Intel, the 8096, has an independent High Speed I/O subsystem which provides the functionality of four to eight 16-bit timers. While this subsystem is designed to provide an integrated approach to measuring and controlling time modulated signals, it is easier to describe as separate input and output units.

## High Speed Input Unit

The purpose of the High Speed Input unit is to allow the measurement of the periods of incoming pulse or frequency modulated inputs with high resolution and minimal software overhead. A block diagram of the hardware used to accomplish this goal is shown in **Figure 1**. The heart of this unit is a programmable change detector which monitors the four I/O pins of the 8096 which are designated as "High Speed Inputs" (HSI.0-HSI.3).

The operating mode of the change detector is controlled by a byte register which can be written as register 3 of the onboard register file. This register has the predeclared name HSI\_MODE in the 8096 assembly language. The register contains a separate field for each of

the four HSI pins. There are two bits in each of these fields and they are encoded as follows:

- 00 Capture every eighth positive transition
- 01 Capture positive transitions only
- 10 Capture negative transitions only
- 11 Capture both positive and negative transitions

It is also possible to disconnect one or more of the HSI pins from the change detector by writing into one of the two I/O control registers. This register, known to the assembly language as IOCO is addressed as register 15H of the on-board register file. HSI pins that

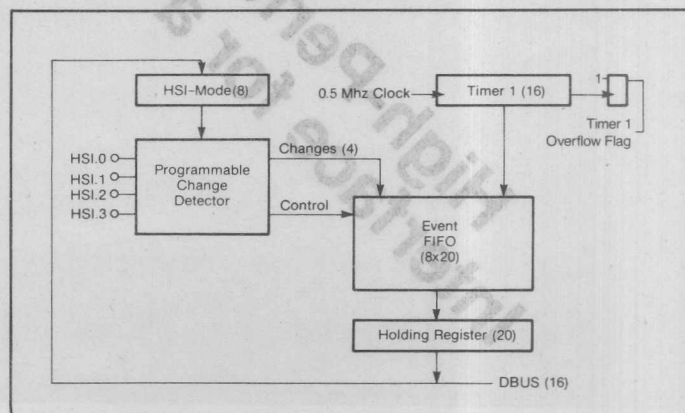


Figure 1: Diagram shows the High-Speed Input Unit which is used to measure incoming pulse or frequency modulated inputs.

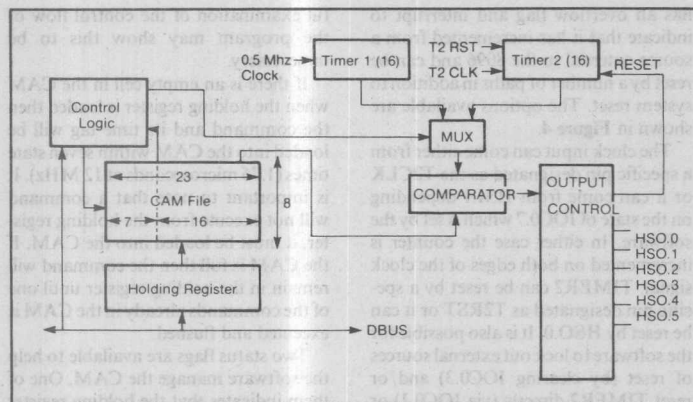


Figure 2: Block diagram of the High Speed Output hardware.

have to be disconnected from the change detector are available for use as normal digital inputs and two of them (HSL2 and HSL3) can be used by the High Speed Output unit.

When a change (or changes) of the required type occurs, four bits of change information, along with the current value of TIMER1, are loaded into a FIFO (first-in, first-out memory). Each set bit in this field indicates that a change occurred on the corresponding input pin.

The time reference for the HSI unit is TIMER1, a sixteen bit counter which is incremented every eight state times by the CPU clock. With a 12 MHz crystal this gives a resolution of 2.0 microsec-

onds. TIMER1 is cleared by reset and then starts incrementing. It cannot be written to by the software but can be read as a sixteen bit word at any time. When its count goes from all ones to zero a flag is set and an interrupt generated. The software can use this flag and/or interrupt to extend the measurement range of the HSI unit.

The FIFO that is used to store the change and time information is eight levels deep (including the holding register) and 20 bits wide. The oldest entry in the FIFO is placed in the holding register. When the holding register is read then the next oldest entry will drop into it and another cell of the FIFO will become available for input data. An

interrupt can be generated either when one or more entries exist in the FIFO or when seven or more entries exist. The choice is made by the software by setting a bit in I/O control register 1 (IOC1).

The 8096 only supports byte and word operands for most operations. The holding register is 20 bits wide hence the holding register is broken down into two registers. The 16-bit time field is read as a word register and is known as HSLTIME to the assembler. The change information is read as an eight-bit byte known as HSLSTATUS. The four extra bits in this byte are used to report the state of the HSI pins at the time the register is read (not at the time the reported change occurred). The holding register is cleared after the HSLTIME is read so that HSLSTATUS can be read at any time to monitor the actual state of the HSI pins without losing data from the FIFO.

### High Speed Output Unit

The High Speed Output unit serves the output requirements of the system in the same way as the HSI unit serves the input. It allows the generation of pulse and frequency modulated signals with high resolution and minimal software overhead. It can also be used to generate time delays for the operating software and to trigger the A/D converter at precise time intervals for signal processing algorithms. A block diagram of the HSO hardware is shown in Figure 2.

The HSO unit is driven by a Content Addressable Memory (CAM) which is 23 bits wide and eight levels deep. The

(continued on page 120)

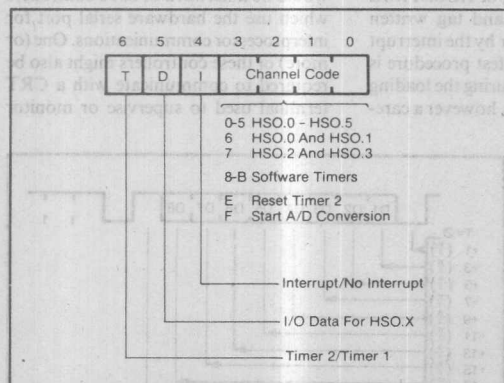


Figure 3: Diagram shows format of Command Tag. The lower four bits specify the basic operation and the remaining three bits are options to the basic operation.

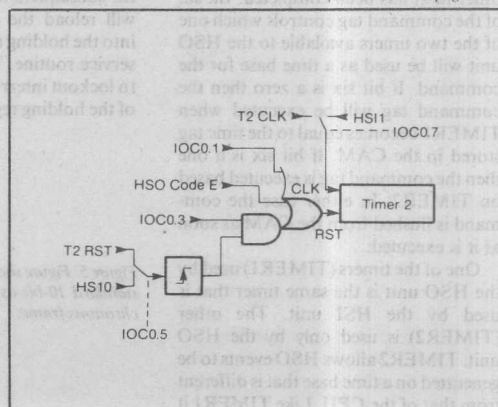


Figure 4: Figure shows the clock and reset options of TIMER2 used by the High Speed Output unit.

23 bits are broken into a 16 bit time tag and a seven bit command tag. The command tag tells it when to do it. The format of the command tag is shown in Figure 3. The lower four bits of the tag specify the basic operation and the remaining three bits specify options to the basic operation. The basic operations supported are:

- Write to one of the six pins controlled by the HSO unit (HSO.0-HSO.5).
- Write to HSO.0 and HSO.1 with a single command.
- Write to HSO.2 and HSO.3 with a single command.
- Set one of four software timer flags.
- Reset Timer 2.
- Trigger an A/D conversion.

If an operation on an HSO pin is specified, then the value to be written to the pin is taken from bit five of the command tag. Note that if two HSO pins are to be modified with the same command then both will be set to the same state. Bit five of the command tag is ignored for the other HSO operations. Bit four of the command tag enables the generation of an interrupt which occurs when the command is executed.

There are two interrupts generated by the HSO unit. One of them indicates that an operation involving a HSO pin has occurred, and the other is used to signal that one of the internal HSO functions (such as setting a software timer flag) has been completed. Bit six of the command tag controls which one of the two timers available to the HSO unit will be used as a time base for the command. If bit six is a zero then the command tag will be executed when TIMER1 becomes equal to the time tag stored in the CAM. If bit six is a one then the command tag is executed based on TIMER2. In either case the command is flushed from the CAM as soon as it is executed.

One of the timers (TIMER1) used by the HSO unit is the same timer that is used by the HSI unit. The other (TIMER2) is used only by the HSO unit. TIMER2 allows HSO events to be generated on a time base that is different from that of the CPU. Like TIMER1 it is a 16 bit counter that can be read but not written to by the software. It also

has an overflow flag and interrupt to indicate that it has incremented from a source external to the 8096 and can be reset by a number of paths in addition to system reset. The options available are shown in Figure 4.

The clock input can come either from a specific pin designated as the T2CLK or it can come from HSI.1 depending on the state of IOC0.7 which is set by the software. In either case the counter is incremented on both edges of the clock signal. TIMER2 can be reset by a specific pin designated as T2RST or it can be reset by HSO.0. It is also possible for the software to lock out external sources of reset (by clearing IOC0.3) and/or reset TIMER2 directly (via IOC0.1) or indirectly via a command stored in the CAM. Note that this last possibility allows TIMER2 to be configured as a modulation counter since the software can command the HSO unit to clear TIMER2 when it reaches a given value.

Commands are loaded into the CAM from the 23 bit wide holding register which, like the holding register for the HSI unit, is actually made up of a byte register (HSO\_COMMAND) which stores the command tag and a word register is considered loaded after HSO\_TIME is loaded so the software must always load HSO\_COMMAND and then load HSO\_TIME.

The software must also ensure that the loading of the two registers is not interrupted by an interrupt service routine which uses the HSO unit. If such an interrupt occurs immediately following the loading of HSO\_COMMAND then the subsequent loading of HSO\_TIME will reload the command tag written into the holding register by the interrupt service routine. The safest procedure is to lockout interrupts during the loading of the holding registers, however a care-

ful examination of the control flow of the program may show this to be unnecessary.

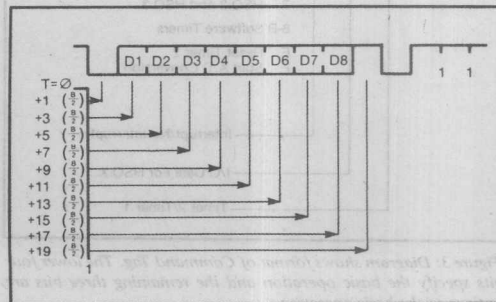
If there is an empty cell in the CAM when the holding register is loaded then the command and its time tag will be loaded into the CAM within seven state times (1.75 microseconds at 12 MHz). It is important to note that a command will not execute from the holding register, it must be loaded into the CAM. If the CAM is full then the command will remain in the holding register until one of the commands already in the CAM is executed and flushed.

Two status flags are available to help the software manage the CAM. One of them indicates that the holding register is full or the CAM is full. Once a command is loaded into the CAM it cannot be read or overwritten, it can only be flushed after it is executed. To support those situations where the software wishes to cancel a command after it has been loaded, the HSO unit is configured so that two operations to a HSO pin which cancel each other will not effect the setting of the pin if they are executed with identical time tags.

### Application Example

Since the 8096 incorporates a full duplex asynchronous serial port in its hardware it may seem strange that one would want to implement a software driven serial port using the high speed I/O features. There are, however, many useful configurations of microcontroller systems which in fact require more than a single serial port. An obvious example would be a network of 8096 controllers which use the hardware serial port for interprocessor communications. One (or more) of these controllers might also be required to communicate with a CRT terminal used to supervise or monitor

Figure 5: Figure shows standard 10-bit asynchronous frame.



---

*The serial output process is simpler than the receive process because there is no need to synchronize with the outside world.*

---

the system. Another example would be a simple CRT terminal design based on an 8096 which needs one serial port for communication and another for driving a slave printer. It may also be true that this is, in fact, a strange requirement. In any case it is an excellent example to show how the high speed I/O features of the 8096 might be used.

The objective is to add a software driven asynchronous serial port to the 8096 that provides full duplex serial communication at 2400 baud. A standard frame consisting of a START bit, eight data bits and a STOP bit will be assumed. A high speed input pin (HSI.0) will be used for received data and a high speed output pin (HSI.0) for transmit data.

A standard 10-bit asynchronous frame is shown in Figure 5. The figure also shows the points in time where the receive process must sample the incom-

ing data stream and take some action. The first timing point (labeled T=0) is the leading edge of the start bit, the accurate sensing of this edge is important because all subsequent sample times are relative to this edge. This event also places the highest burden on the sampling algorithm because it can occur at any point in time. The rest of the sampling events occur at some multiple of one-half a bit period relative to the edge of the start bit. The diagram uses the symbol B to represent a bit period.

At the second sample, which occurs half way through the start bit, the data must be checked to make sure it is still a SPACE. If it is not, a noise pulse has caused a false start and the receive process must be reinitialized. The next eight samples are used to shift in the serial data stream. The last sample, which occurs 19 one-half bit times after the leading edge of the start bit, is used to verify that the stop bit is valid (i.e. it is in the MARK state). If it is not the a framing error must be reported since it is likely that the receiver is not properly synchronized with the transmitter.

The HSI unit is an ideal mechanism for detecting the leading edge of the start bit. All that needs to be done is to set the mode register to detect negative going edges on HSI.0.

The software timer interrupt service routine implements a simple state machine based on the variable count. The routine also arranges for the next sample by issuing a command to the HSO unit to generate another software

timer interrupt at the appropriate time. This is done in all states unless the reception of the character is complete or a false START bit has been detected. Under these conditions the receive process must be reinitialized by enabling HSI.0 into the event FIFO (by setting IOC0.0) instead of retriggering the software timer.

The serial output process is simpler than the receive process because there is no need to synchronize with the outside world. A transmission can be started at any time by setting the TxD line to a space for one bit time to form the START bit. Following the START bit are the eight data bits and the STOP bit. The HSO interrupt service routine can be used to transmit the data and the stop bit but the transmit process must be initialized.

The only real complication in the HSO interrupt service routine is that there are no flags available in the 8096 which indicate which of the HSO outputs caused the interrupt. In many systems this does not represent a problem because the HSO unit can be treated as a write only device. It is given commands which are to be executed at the proper time but no feedback is required to indicate when the proper time has been reached. In this case, however, the feedback is required since the CAM isn't deep enough to hold all of the transitions required for a character. Even if it were big enough it is unlikely that so many CAM locations would be dedicated to serial output. □



## The serial output process is simpler than the receive process because there is no need to synchronize with the outside world.

The system. Another example would be a simple CMT terminal design based on an 8096 which needs one serial port for communication and another for driving a slave printer. It may also be true that this is in fact a strange requirement in any case. It is an excellent example to show how the high-speed I/O features of the 8096 might be used.

The objective is to add a software-driven asynchronous serial port to the 8096 that provides full duplex serial communication at 2400 baud. A standard frame consisting of a START bit, eight data bits and a STOP bit will be assumed. A high-speed input pin (H210) will be used for received data and a high-speed output pin (H210) for transmit data.

A standard 10-bit asynchronous frame is shown in Figure 2. The figure also shows the points in time where the receive process must sample the incoming

data stream and how such a device would sample the data stream. The first sampling point (labeled T-0) is the leading edge of the start bit. The accuracy of this edge is important because all subsequent sample times are relative to this edge. This event also places the highest burden on the sampling algorithm because it can occur at any point in time. The rest of the sampling events occur at some multiple of one-half a bit period relative to the edge of the start bit. The diagram uses the symbol R to represent a bit period.

At the second sample, which occurs half way through the start bit, the data must be checked to make sure it is still a SPACE. If it is not, a noise pulse has caused a false start and the receive process must be reinitialized. The next eight samples are used to shift in the serial data stream. The last sample, which occurs 19 one-half bit times after the leading edge of the start bit, is used to verify that the stop bit is valid (i.e. it is the MARK state). If it is not the transmitting error must be reported since it is likely that the receiver is not properly synchronized with the transmitter.

The H21 unit is an ideal mechanism for detecting the leading edge of the start bit. All that needs to be done is to set the mode register to detect negative-going edges on H210.

The software timer interrupt service routine implements a simple state machine based on the variable count. The routine also arranges for the next sample by issuing a command to the H20 unit to generate another software

timer interrupt at the appropriate time. This is done in all states unless the reception of the character is complete or a false START bit has been detected. Under these conditions the receive process must be reinitialized by changing H210 into the event FIFO (by setting IOC00) instead of retriggering the software timer.

The serial output process is simpler than the receive process because there is no need to synchronize with the outside world. A transmission can be started at any time by setting the TXD line to a space for one bit time to form the START bit. Following the START bit are the eight data bits and the STOP bit. The H20 interrupt service routine can be used to transmit the data and the stop bit but the transmit process must be initialized.

The only real complication in the H20 interrupt service routine is that there are no flags available in the 8096 which indicate which of the H20 outputs caused the interrupt. In many systems this does not represent a problem because the H20 unit can be treated as a write-only device. It is given commands which are to be executed at the proper time but no feedback is required to indicate when the proper time has been reached. In this case, however, the feedback is required since the CAM unit deep enough to hold all of the transmit data. Even if it is unlikely that so many CAM locations would be defined, the output would be defined.



7

MCS®-51 Architecture





## MCS<sup>®</sup>-51 ARCHITECTURE

The newest MCS-51 members, the 8032 and 8052, have more on-chip memory and an additional 16-bit timer/counter. The new timer can be used as a timer, a counter, or to generate baud rates for the serial port. As a timer/counter, it operates in either a 16-bit auto-reload mode or a 16-bit "capture" mode. This new feature is described in Section 7.6.2.

Pinouts are shown in the individual data sheets and on the inside back cover of this handbook.

Table 1. MCS<sup>®</sup>-51 Family Members

PART	TECHNOLOGY	ON-CHIP PROGRAM MEMORY	ON-CHIP DATA MEMORY
8051	HMOS	4K — ROM	128
8031	HMOS	NONE	128
8751H	HMOS I	4K — EPROM	128
80C51	CHMOS	4K — ROM	128
80C31	CHMOS	NONE	128
8052	HMOS	8K — ROM	256
8032	HMOS	NONE	256

The major MCS<sup>®</sup>-51 features are:

- 8-Bit CPU
- On-Chip oscillator and clock circuitry
- 32 I/O lines
- 64K address space for external data memory
- 64K address space for external program memory
- Two 16-bit timer/counters (three on 8032/8052)
- A five-source interrupt structure (six sources on 8032/8052) with two priority levels
- Full duplex serial port
- Boolean processor

### 7.1 MEMORY ORGANIZATION

The 8051 has separate address spaces for Program Memory and Data Memory. The Program Memory can be up to 64K bytes long. The lower 4K (8K for 8052) may reside on-chip. The Data Memory can consist of up to 64K bytes of off-chip RAM, in addition to which it includes 128 bytes of on-chip RAM (256 bytes for the 8052), plus a number of "SFRs" (Special Function Registers) as listed below.

Symbol	Name	Address
*ACC	Accumulator	0E0H
*B	B Register	0F0H
*PSW	Program Status Word	0D0H
SP	Stack Pointer	81H
DPTR	Data Pointer (consisting of DPH and DPL)	83H 82H
*P0	Port 0	80H
*P1	Port 1	90H

Symbol	Name	Address
*P2	Port 2	0A0H
*P3	Port 3	0B0H
*IP	Interrupt Priority Control	0B8H
*IE	Interrupt Enable Control	0A8H
TMOD	Timer/Counter Mode Control	89H
*TCON	Timer/Counter Control	88H
+ *T2CON	Timer/Counter 2 Control	0C8H
TH0	Timer/Counter 0 (high byte)	8CH
TL0	Timer/Counter 0 (low byte)	8AH
TH1	Timer/Counter 1 (high byte)	8DH
TL1	Timer/Counter 1 (low byte)	8BH
+ TH2	Timer/Counter 2 (high byte)	0CDH
+ TL2	Timer/Counter 2 (low byte)	0CCH
+ RCAP2H	Timer/Counter 2 Capture Register (high byte)	0CBH
+ RCAP2L	Timer/Counter 2 Capture Register (low byte)	0CAH
*SCON	Serial Control	98H
SBUF	Serial Data Buff	99H
PCON	Power Control	87H

The SFRs marked with an asterisk (\*) are both bit- and byte-addressable. The SFRs marked with a plus sign (+) are present in the 8052 only. The functions of the SFRs are described as follows.

### ACCUMULATOR

ACC is the Accumulator register. The mnemonics for accumulator-specific instructions, however, refer to the accumulator simply as A.

### B REGISTER

The B register is used during multiply and divide operations. For other instructions it can be treated as another scratch pad register.

### PROGRAM STATUS WORD

The PSW register contains program status information as detailed in Figure 7-2.

### STACK POINTER

The Stack Pointer register is 8 bits wide. It is incremented before data is stored during PUSH and CALL executions. While the stack may reside anywhere in on-chip RAM,

the Stack Pointer is initialized to 07H after a reset. This causes the stack to begin at location 08H.

## DATA POINTER

The Data Pointer (DPTR) consists of a high byte (DPH) and a low byte (DPL). Its intended function is to hold a 16-bit address. It may be manipulated as a 16-bit register or as two independent 8-bit registers.

## PORTS 0 to 3

P0, P1, P2 and P3 are the SFR latches of Ports 0, 1, 2 and 3, respectively.

## SERIAL DATA BUFFER

The Serial Data Buffer is actually two separate registers, a transmit buffer and a receive buffer register. When data is moved to SBUF, it goes to the transmit buffer where it is held for serial transmission. (Moving a byte to SBUF is what initiates the transmission.) When data is moved from SBUF, it comes from the receive buffer.

## TIMER REGISTERS

Register pairs (TH0, TL0), (TH1, TL1), and (TH2, TL2) are the 16-bit counting registers for Timer/Counters 0, 1, and 2, respectively.

## CAPTURE REGISTERS

The register pair (RCAP2H, RCAP2L) are the capture registers for the Timer 2 "capture mode." In this mode, in response to a transition at the 8052's T2EX pin, TH2 and TL2 are copied into RCAP2H and RCAP2L. Timer

2 also has a 16-bit auto-reload mode, and RCAP2H and RCAP2L hold the reload value for this mode. More about Timer 2's features in Section 7.6.2.

## CONTROL REGISTERS

Special Function Registers IP, IE, TMOD, TCON, T2CON, SCON, and PCON contain control and status bits for the interrupt system, the timer/counters, and the serial port. They are described in later sections.

## 7.2 OSCILLATOR AND CLOCK CIRCUIT

XTAL1 and XTAL2 are the input and output of a single-stage on-chip inverter, which can be configured with off-chip components as a Pierce oscillator, as shown in Figure 7-3. The on-chip circuitry, and selection of off-chip components to configure the oscillator are discussed in Section

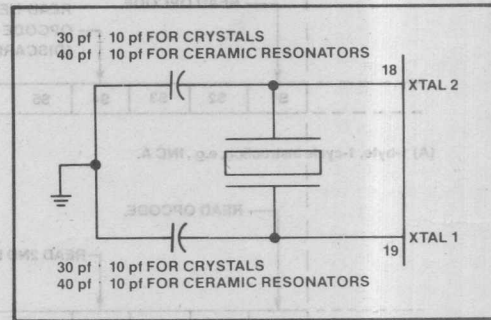


Figure 7-3. Crystal/Ceramic Resonator Oscillator

(MSB)				(LSB)			
CY	AC	F0	RS1	RS0	OV	—	P
Symbol	Position	Name and Significance		Symbol	Position	Name and Significance	
CY	PSW.7	Carry flag.		OV	PSW.2	Overflow flag.	
AC	PSW.6	Auxiliary Carry flag. (For BCD operations.)		—	PSW.1	(reserved)	
F0	PSW.5	Flag 0 (Available to the user for general purposes.)		P	PSW.0	Parity flag. Set/cleared by hardware each instruction cycle to indicate an odd/even number of "one" bits in the accumulator, i.e., even parity.	
RS1	PSW.4	Register bank Select control bits 1 & 0. Set/cleared by software to determine working register bank (see Note).		Note— the contents of (RS1, RS0) enable the working register banks as follows:			
RS0	PSW.3						
				(0.0)—Bank 0 (00H-07H)			
				(0.1)—Bank 1 (08H-0FH)			
				(1.0)—Bank 2 (10H-17H)			
				(1.1)—Bank 3 (18H-1FH)			

Figure 7-2. PSW: Program Status Word Register

## MCS<sup>®</sup>-51 ARCHITECTURE

7.13. A more detailed discussion will be found in Application Note AP-155, "Oscillators for Microcontrollers," which is included in this manual.

The oscillator, in any case, drives the internal clock generator. The clock generator provides the internal clocking signals to the chip. The internal clocking signals are at half the oscillator frequency, and define the internal

phases, states, and machine cycles, which are described in the next section.

### 7.3 CPU TIMING

A machine cycle consists of 6 states (12 oscillator periods). Each state is divided into a Phase 1 half, during which the Phase 1 clock is active, and a Phase 2 half,

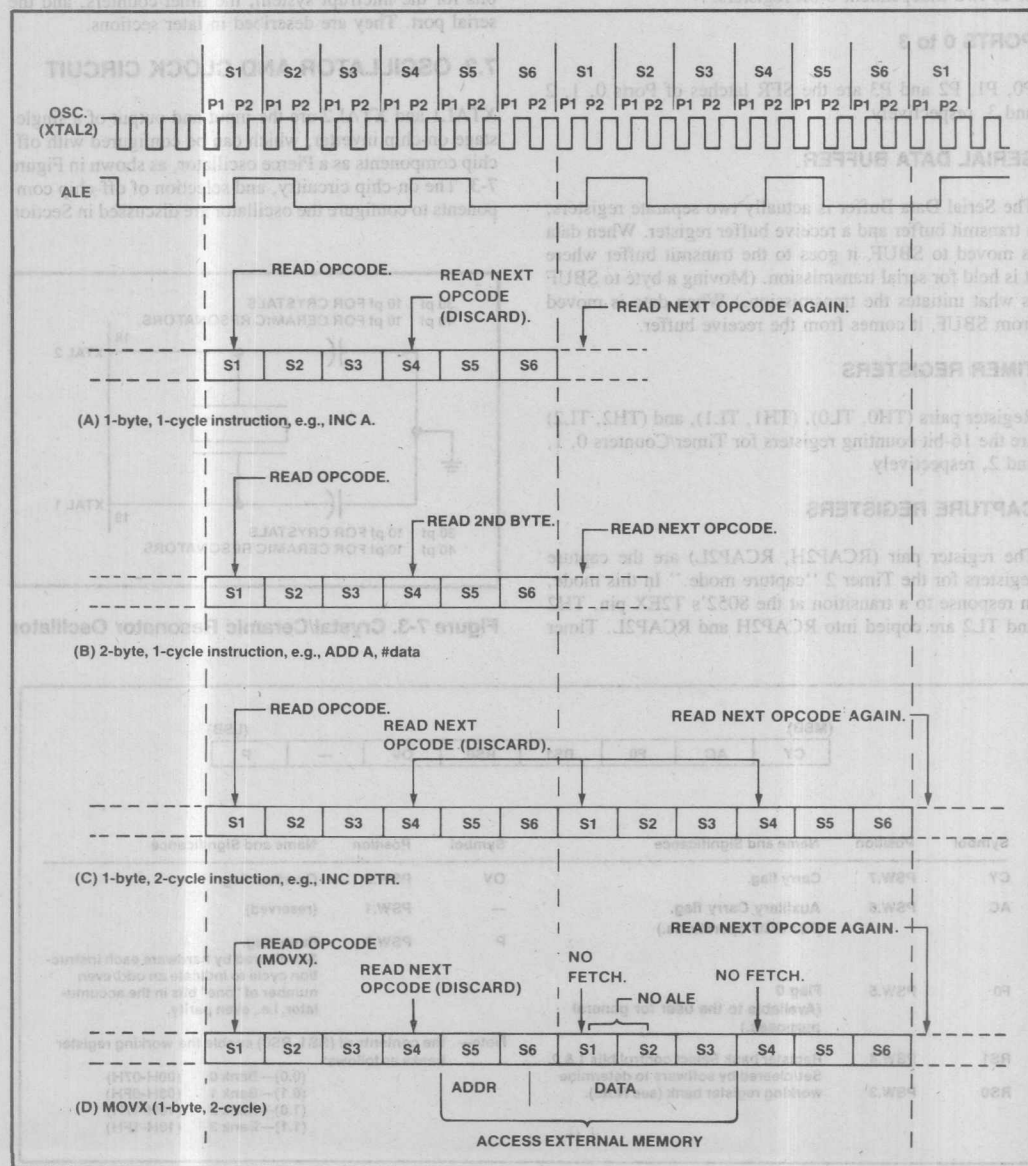


Figure 7-4. 8051 Fetch/Execute Sequences

during which the Phase 2 clock is active. Thus, a machine cycle consists of 12 oscillator periods, numbered S1P1 (State 1, Phase 1), through S6P2 (State 6, Phase 2). Each phase lasts for one oscillator period. Each state lasts for two oscillator periods. Typically, arithmetic and logical operations take place during Phase 1 and internal register-to-register transfers take place during Phase 2.

The diagrams in Figure 7-4 show the fetch/execute timing referenced to the internal states and phases. Since these internal clock signals are not user accessible, the XTAL2 oscillator signal and the ALE (Address Latch Enable) signal are shown for external reference. ALE is normally activated twice during each machine cycle: once during S1P2 and S2P1, and again during S4P2 and S5P1.

Execution of a one-cycle instruction begins at S1P2, when the opcode is latched into the Instruction Register. If it is a two-byte instruction, the second byte is read during S4 of the same machine cycle. If it is a one-byte instruction, there is still a fetch at S4, but the byte read (which would be the next opcode), is ignored, and the Program Counter is not incremented. In any case, execution is complete at

the end of S6P2. Figures 7-4A and 7-4B show the timing for a 1-byte, 1-cycle instruction and for a 2-byte, 1-cycle instruction.

Most 8051 instructions execute in one cycle. MUL (multiply) and DIV (divide) are the only instructions that take more than two cycles to complete. They take four cycles:

Normally, two code bytes are fetched from Program Memory during every machine cycle. The only exception to this is when a MOVX instruction is executed. MOVX is a 1-byte 2-cycle instruction that accesses external Data Memory. During a MOVX, two fetches are skipped while the external Data Memory is being addressed and strobed. Figures 7-4C and 7-4D show the timing for a normal 1-byte, 2-cycle instruction and for a MOVX instruction.

## 7.4 PORT STRUCTURES AND OPERATION

All four ports in the 8051 are bidirectional. Each consists of a latch (Special Function Registers P0 through P3), an output driver, and an input buffer.

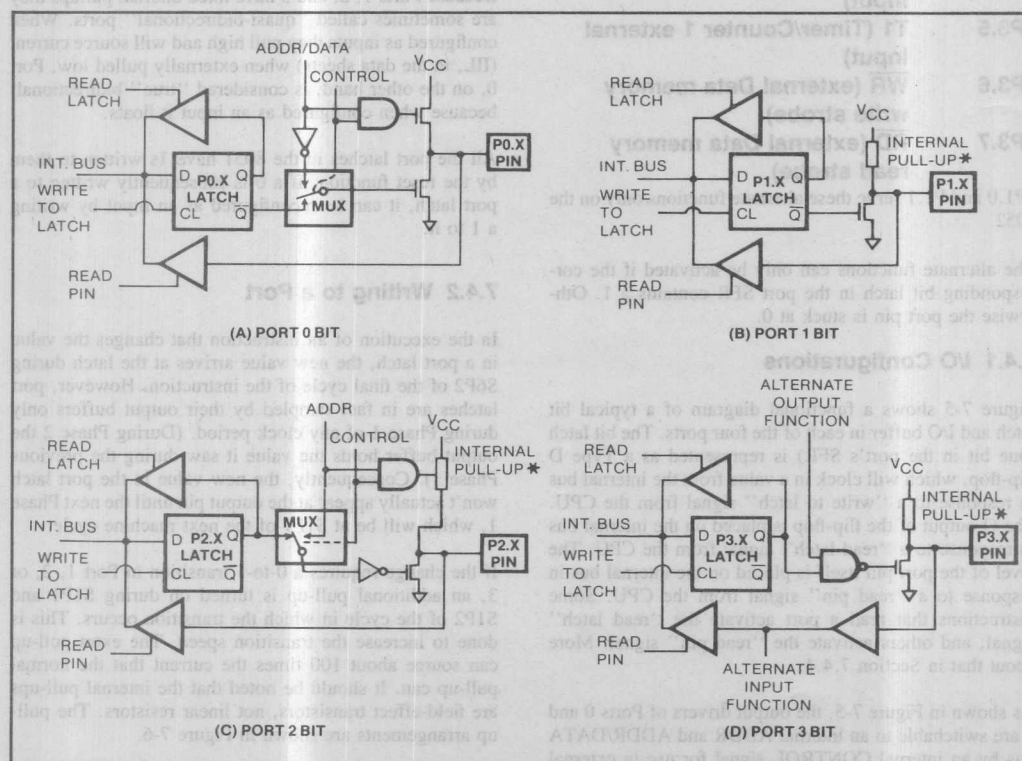


Figure 7-5. 8051 Port Bit Latches and I/O Buffers

\*See Figure 7-6 for details of the internal pullup.



The output drivers of Ports 0 and 2, and the input buffers of Port 0, are used in accesses to external memory. In this application, Port 0 outputs the low byte of the external memory address, time-multiplexed with the byte being written or read. Port 2 outputs the high byte of the external memory address when the address is 16 bits wide. Otherwise the Port 2 pins continue to emit the P2 SFR content.

All the Port 3 pins, and (in the 8052) two Port 1 pins are multifunctional. They are not only port pins, but also serve the functions of various special features as listed below:

#### PORT PIN ALTERNATE FUNCTION

*P1.0	T2 (Timer/Counter 2 external input)
*P1.1	T2EX (Timer/Counter 2 capture/reload trigger)
P3.0	RXD (serial input port)
P3.1	TXD (serial output port)
P3.2	INT0 (external interrupt)
P3.3	INT1 (external interrupt)
P3.4	T0 (Timer/Counter 0 external input)
P3.5	T1 (Timer/Counter 1 external input)
P3.6	WR (external Data memory write strobe)
P3.7	RD (external Data memory read strobe)

\*P1.0 and P1.1 serve these alternate functions only on the 8052.

The alternate functions can only be activated if the corresponding bit latch in the port SFR contains a 1. Otherwise the port pin is stuck at 0.

#### 7.4.1 I/O Configurations

Figure 7-5 shows a functional diagram of a typical bit latch and I/O buffer in each of the four ports. The bit latch (one bit in the port's SFR) is represented as a Type D flip-flop, which will clock in a value from the internal bus in response to a "write to latch" signal from the CPU. The Q output of the flip-flop is placed on the internal bus in response to a "read latch" signal from the CPU. The level of the port pin itself is placed on the internal bus in response to a "read pin" signal from the CPU. Some instructions that read a port activate the "read latch" signal, and others activate the "read pin" signal. More about that in Section 7.4.4.

As shown in Figure 7-5, the output drivers of Ports 0 and 2 are switchable to an internal ADDR and ADDR/DATA bus by an internal CONTROL signal for use in external memory accesses. During external memory accesses, the P2 SFR remains unchanged, but the P0 SFR gets 1s written to it.

Also shown in Figure 7-5, is that if a P3 bit latch contains a 1, then the output level is controlled by the signal labeled "alternate output function." The actual P3.X pin level is always available to the pin's alternate input function, if any.

Ports 1, 2, and 3 have internal pull-ups. Port 0 has open-drain outputs. Each I/O line can be independently used as an input or an output. (Ports 0 and 2 may not be used as general purpose I/O when being used as the ADDR/DATA BUS). To be used as an input, the port bit latch must contain a 1, which turns off the output driver FET. Then, for Ports 1, 2, and 3, the pin is pulled high by the internal pull-up, but can be pulled low by an external source.

Port 0 differs in not having internal pullups. The pullup FET in the P0 output driver (see Figure 7-5A) is used only when the Port is emitting 1s during external memory accesses. Otherwise the pullup FET is off. Consequently P0 lines that are being used as output port lines are open drain. Writing a 1 to the bit latch leaves both output FETs off, so the pin floats. In that condition it can be used as a high-impedance input.

Because Ports 1, 2, and 3 have fixed internal pullups they are sometimes called "quasi-bidirectional" ports. When configured as inputs they pull high and will source current (IIL, in the data sheets) when externally pulled low. Port 0, on the other hand, is considered "true" bidirectional, because when configured as an input it floats.

All the port latches in the 8051 have 1s written to them by the reset function. If a 0 is subsequently written to a port latch, it can be reconfigured as an input by writing a 1 to it.

#### 7.4.2 Writing to a Port

In the execution of an instruction that changes the value in a port latch, the new value arrives at the latch during S6P2 of the final cycle of the instruction. However, port latches are in fact sampled by their output buffers only during Phase 1 of any clock period. (During Phase 2 the output buffer holds the value it saw during the previous Phase 1). Consequently, the new value in the port latch won't actually appear at the output pin until the next Phase 1, which will be at S1P1 of the next machine cycle.

If the change requires a 0-to-1 transition in Port 1, 2, or 3, an additional pull-up is turned on during S1P1 and S1P2 of the cycle in which the transition occurs. This is done to increase the transition speed. The extra pull-up can source about 100 times the current that the normal pull-up can. It should be noted that the internal pull-ups are field-effect transistors, not linear resistors. The pull-up arrangements are shown in Figure 7-6.

In HMOS versions of the 8051, the fixed part of the pull-up is a depletion-mode transistor with the gate wired to the source. This transistor will allow the pin to source

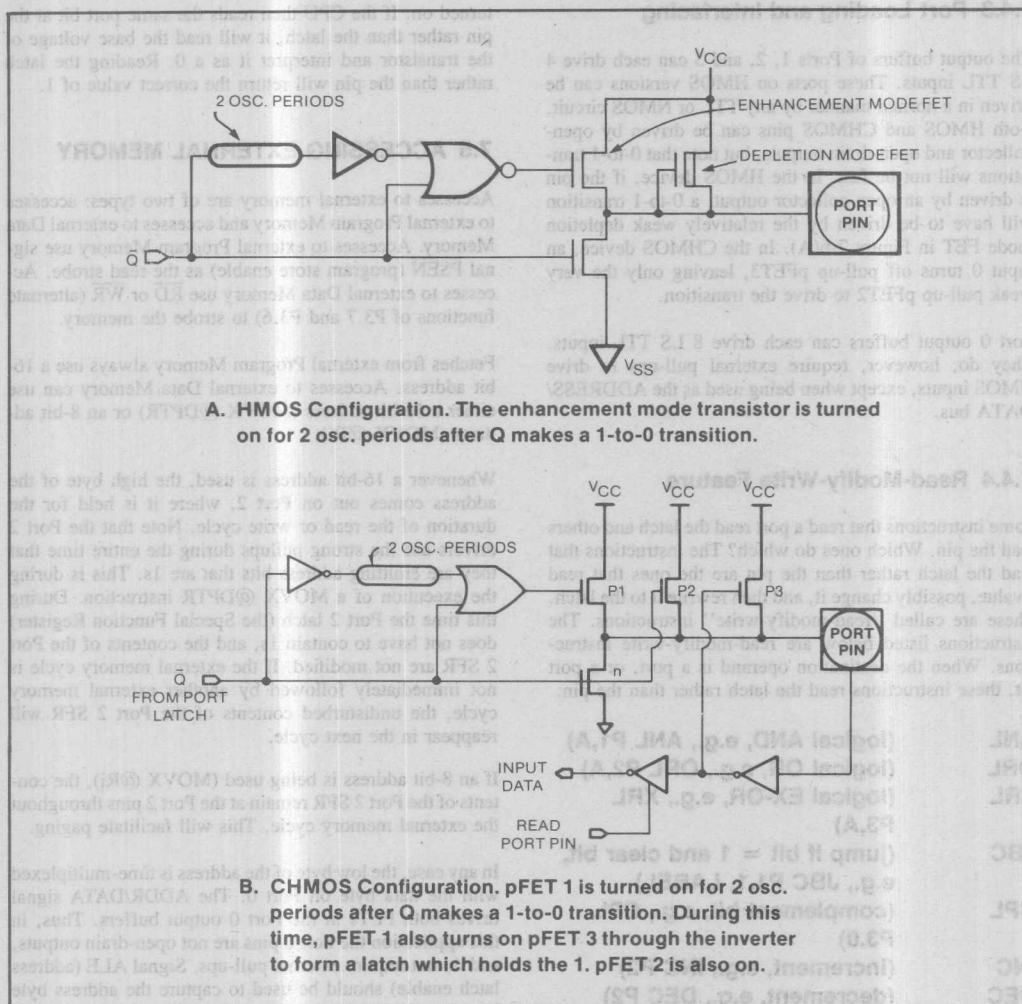


Figure 7-6. Ports 1 and 3 HMOS and CHMOS Internal Pull-up Configurations.

Port 2 is similar except that it holds the strong pullup on while emitting 1s that are address bits.

(See text, "Accessing External Memory.")

about 0.25 mA when shorted to ground. In parallel with the fixed pull-up is an enhancement-mode transistor, which is activated during S1 whenever the port bit does a 0-to-1 transition. During this interval, if the port pin is shorted to ground, this extra transistor will allow the pin to source an additional 30 mA.

In the CHMOS versions, the pull-up consists of three pFETs. It should be noted that an n-channel FET (nFET) is turned on when a logical 1 is applied to its gate, and is turned off when a logical 0 is applied to its gate. A p-channel FET (pFET) is the opposite: it is on when its gate sees a 0, and off when its gate sees a 1.

pFET 1 in Figure 7-6B is the transistor that is turned on for 2 oscillator periods after a 0-to-1 transition in the port latch. While it's on, it turns on pFET 3 (a weak pull-up), through the inverter. This inverter and pFET form a latch which holds the 1.

Note that if the pin is emitting a 1, a negative glitch on the pin from some external source can turn off pFET 3, causing the pin to go into a float state, pFET 2 is a very weak pull-up which is on whenever the nFET is off, in traditional CMOS style. It's only about 1/10 the strength of pFET3. Its function is to restore a 1 to the pin in the event the pin had a 1 and lost it to a glitch.

### 7.4.3 Port Loading and Interfacing

The output buffers of Ports 1, 2, and 3 can each drive 4 LS TTL inputs. These ports on HMOS versions can be driven in a normal manner by any TTL or NMOS circuit. Both HMOS and CHMOS pins can be driven by open-collector and open-drain outputs, but note that 0-to-1 transitions will not be fast. In the HMOS device, if the pin is driven by an open collector output, a 0-to-1 transition will have to be driven by the relatively weak depletion mode FET in Figure 7-6(A). In the CHMOS device, an input 0 turns off pull-up pFET3, leaving only the very weak pull-up pFET2 to drive the transition.

Port 0 output buffers can each drive 8 LS TTL inputs. They do, however, require external pull-ups to drive NMOS inputs, except when being used as the ADDRESS/DATA bus.

### 7.4.4 Read-Modify-Write Feature

Some instructions that read a port read the latch and others read the pin. Which ones do which? The instructions that read the latch rather than the pin are the ones that read a value, possibly change it, and then rewrite it to the latch. These are called "read-modify-write" instructions. The instructions listed below are read-modify-write instructions. When the destination operand is a port, or a port bit, these instructions read the latch rather than the pin:

<b>ANL</b>	(logical AND, e.g., ANL P1,A)
<b>ORL</b>	(logical OR, e.g., ORL P2,A)
<b>XRL</b>	(logical EX-OR, e.g., XRL P3,A)
<b>JBC</b>	(jump if bit = 1 and clear bit, e.g., JBC P1.1, LABEL)
<b>CPL</b>	(complement bit, e.g., CPL P3.0)
<b>INC</b>	(increment, e.g., INC P2)
<b>DEC</b>	(decrement, e.g., DEC P2)
<b>DJNZ</b>	(decrement and jump if not zero, e.g., DJNZ P3, LABEL)
<b>MOV PX,Y,C</b>	(move carry bit to bit Y of Port X)
<b>CLR PX.Y</b>	(clear bit Y of Port X)
<b>SET PX.Y</b>	(set bit Y of Port X)

It is not obvious that the last three instructions in this list are read-modify-write instructions, but they are. They read the port byte, all 8 bits, modify the addressed bit, then write the new byte back to the latch.

The reason that read-modify-write instructions are directed to the latch rather than the pin is to avoid a possible misinterpretation of the voltage level at the pin. For example, a port bit might be used to drive the base of a transistor. When a 1 is written to the bit, the transistor is

turned on. If the CPU then reads the same port bit at the pin rather than the latch, it will read the base voltage of the transistor and interpret it as a 0. Reading the latch rather than the pin will return the correct value of 1.

## 7.5 ACCESSING EXTERNAL MEMORY

Accesses to external memory are of two types: accesses to external Program Memory and accesses to external Data Memory. Accesses to external Program Memory use signal  $\overline{\text{PSEN}}$  (program store enable) as the read strobe. Accesses to external Data Memory use  $\overline{\text{RD}}$  or  $\overline{\text{WR}}$  (alternate functions of P3.7 and P3.6) to strobe the memory.

Fetches from external Program Memory always use a 16-bit address. Accesses to external Data Memory can use either a 16-bit address ( $\text{MOVX @DPTR}$ ) or an 8-bit address ( $\text{MOVX @Ri}$ ).

Whenever a 16-bit address is used, the high byte of the address comes out on Port 2, where it is held for the duration of the read or write cycle. Note that the Port 2 drivers use the strong pullups during the entire time that they are emitting address bits that are 1s. This is during the execution of a  $\text{MOVX @DPTR}$  instruction. During this time the Port 2 latch (the Special Function Register) does not have to contain 1s, and the contents of the Port 2 SFR are not modified. If the external memory cycle is not immediately followed by another external memory cycle, the undisturbed contents of the Port 2 SFR will reappear in the next cycle.

If an 8-bit address is being used ( $\text{MOVX @Ri}$ ), the contents of the Port 2 SFR remain at the Port 2 pins throughout the external memory cycle. This will facilitate paging.

In any case, the low byte of the address is time-multiplexed with the data byte on Port 0. The ADDR/DATA signal drives both FETs in the Port 0 output buffers. Thus, in this application the Port 0 pins are not open-drain outputs, and do not require external pull-ups. Signal ALE (address latch enable) should be used to capture the address byte into an external latch. The address byte is valid at the negative transition of ALE. Then, in a write cycle, the data byte to be written appears on Port 0 just before  $\overline{\text{WR}}$  is activated, and remains there until after  $\overline{\text{WR}}$  is deactivated. In a read cycle, the incoming byte is accepted at Port 0 just before the read strobe is deactivated.

During any access to external memory, the CPU writes  $\text{OFFH}$  to the Port 0 latch (the Special Function Register), thus obliterating whatever information the Port 0 SFR may have been holding.

External Program Memory is accessed under two conditions:

- 1) Whenever signal  $\overline{\text{EA}}$  is active; or
- 2) Whenever the program counter (PC) contains a number that is larger than  $\text{0FFFH}$  ( $\text{1FFFH}$  for the 8052).

This requires that the ROMless versions have  $\overline{\text{EA}}$  wired

low to enable the lower 4K (8K for the 8032) program bytes to be fetched from external memory.

When the CPU is executing out of external Program Memory, all 8 bits of Port 2 are dedicated to an output function and may not be used for general purpose I/O. During external program fetches they output the high byte of the PC. During this time the Port 2 drivers use the strong pullups to emit PC bits that are 1s.

### 7.5.1 $\overline{\text{PSEN}}$

The read strobe for external fetches is  $\overline{\text{PSEN}}$ .  $\overline{\text{PSEN}}$  is not activated for internal fetches. When the CPU is accessing external Program Memory,  $\overline{\text{PSEN}}$  is activated twice every cycle (except during a MOVX instruction) whether or not the byte fetched is actually needed for the current instruction. When  $\overline{\text{PSEN}}$  is activated its timing is not the same as  $\overline{\text{RD}}$ . A complete  $\overline{\text{RD}}$  cycle, including activation and deactivation of ALE and  $\overline{\text{RD}}$ , takes 12

oscillator periods. A complete  $\overline{\text{PSEN}}$  cycle, including activation and deactivation of ALE and  $\overline{\text{PSEN}}$ , takes 6 oscillator periods. The execution sequence for these two types of read cycles are shown in Figure 7-7 for comparison.

### 7.5.2 ALE

The main function of ALE is to provide a properly timed signal to latch the low byte of an address from P0 to an external latch during fetches from external Program Memory. For that purpose ALE is activated twice every machine cycle. This activation takes place even when the cycle involves no external fetch. The only time an ALE pulse doesn't come out is during an access to external Data Memory. The first ALE of the second cycle of a MOVX instruction is missing (see Figure 7-7). Consequently, in any system that does not use external Data Memory, ALE is activated at a constant rate of 1/6 the oscillator frequency, and can be used for external clocking or timing purposes.

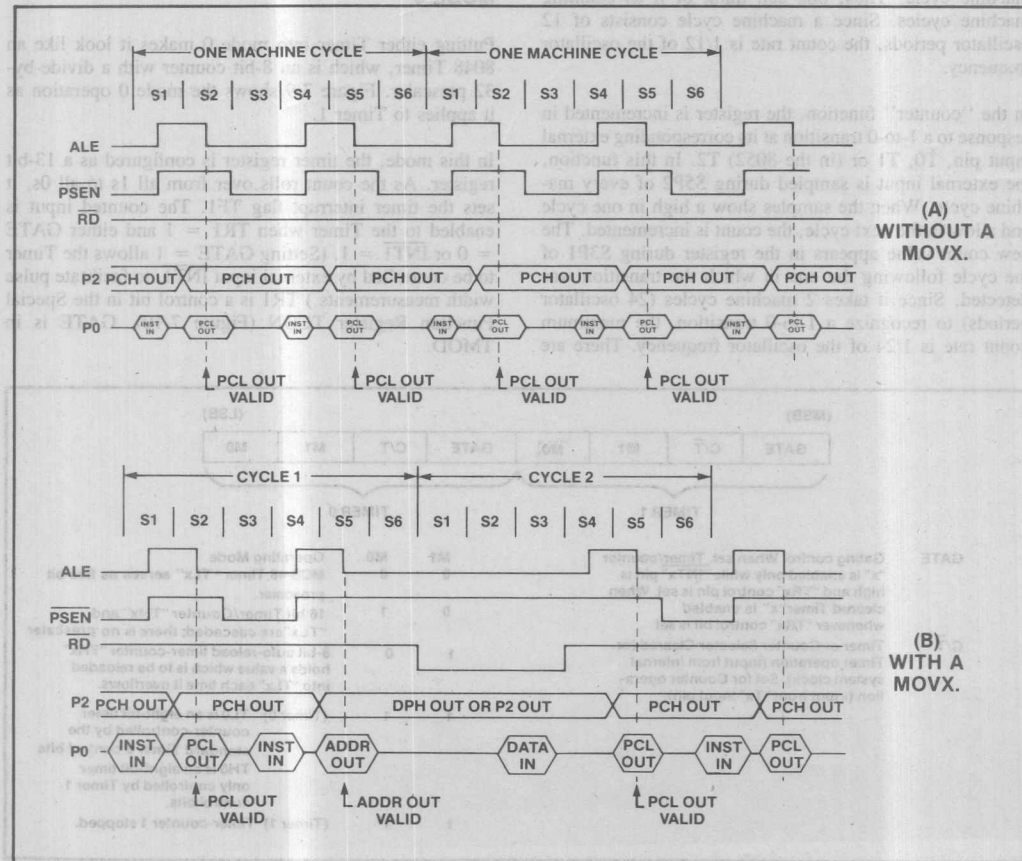


Figure 7-7. External Program Memory Execution



### 7.5.3 Overlapping External Program and Data Memory Spaces

In some applications it is desirable to execute a program from the same physical memory that is being used to store data. In the 8051, the external Program and Data Memory spaces can be combined by ANDing  $\overline{\text{PSEN}}$  and  $\overline{\text{RD}}$ . A positive-logic AND of these two signals produces an active-low read strobe that can be used for the combined physical memory. Since the  $\overline{\text{PSEN}}$  cycle is faster than the  $\overline{\text{RD}}$  cycle, the external memory needs to be fast enough to accommodate the  $\overline{\text{PSEN}}$  cycle.

### 7.6 TIMER/COUNTERS

The 8051 has two 16-bit timer/counter registers: Timer 0 and Timer 1. The 8052 has these two plus one more: Timer 2. All three can be configured to operate either as timers or event counters.

In the "timer" function, the register is incremented every machine cycle. Thus, one can think of it as counting machine cycles. Since a machine cycle consists of 12 oscillator periods, the count rate is 1/12 of the oscillator frequency.

In the "counter" function, the register is incremented in response to a 1-to-0 transition at its corresponding external input pin, T0, T1 or (in the 8052) T2. In this function, the external input is sampled during S5P2 of every machine cycle. When the samples show a high in one cycle and a low in the next cycle, the count is incremented. The new count value appears in the register during S3P1 of the cycle following the one in which the transition was detected. Since it takes 2 machine cycles (24 oscillator periods) to recognize a 1-to-0 transition, the maximum count rate is 1/24 of the oscillator frequency. There are

no restrictions on the duty cycle of the external input signal, but to ensure that a given level is sampled at least once before it changes, it should be held for at least one full machine cycle.

In addition to the "timer" or "counter" selection, Timer 0 and Timer 1 have four operating modes from which to select. Timer 2, in the 8052, has three modes of operation: "capture," "auto-reload" and "baud rate generator."

#### 7.6.1 Timer 0 and Timer 1

These timer/counters are present in both the 8051 and the 8052. The "timer" or "counter" function is selected by control bits C/T in the Special Function Register TMOD (Figure 6-8). These two timer/counters have four operating modes, which are selected by bit-pairs (M1, M0) in TMOD. Modes 0, 1, and 2 are the same for both timer/counters. Mode 3 is different. The four operating modes are described below.

#### MODE 0

Putting either Timer into mode 0 makes it look like an 8048 Timer, which is an 8-bit counter with a divide-by-32 prescaler. Figure 7-9 shows the mode 0 operation as it applies to Timer 1.

In this mode, the timer register is configured as a 13-bit register. As the count rolls over from all 1s to all 0s, it sets the timer interrupt flag TF1. The counted input is enabled to the Timer when TR1 = 1 and either GATE = 0 or INT1 = 1. (Setting GATE = 1 allows the Timer to be controlled by external input INT1, to facilitate pulse width measurements.) TR1 is a control bit in the Special Function Register TCON (Figure 7-10). GATE is in TMOD.

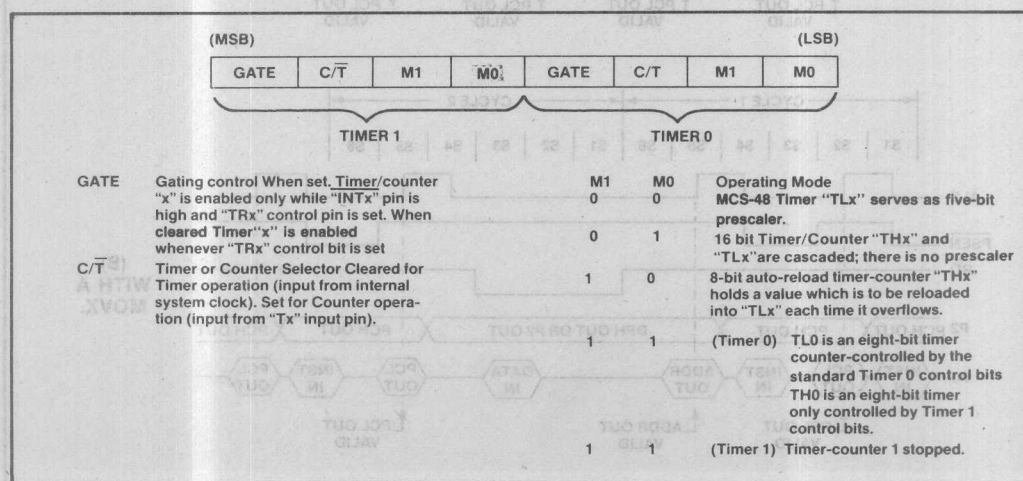


Figure 7-8. TMOD: Timer/Counter Mode Control Register

The 13-bit register consists of all 8 bits of TH1 and the lower 5 bits of TL1. The upper 3 bits of TL1 are indeterminate and should be ignored. Setting the run flag (TR1) does not clear the registers.

Mode 0 operation is the same for Timer 0 as for Timer 1. Substitute TR0, TF0 and INT0 for the corresponding Timer 1 signals in Figure 7-9. There are two different GATE bits, one for Timer 1 (TMOD.7) and one for Timer 0 (TMOD.3).

#### MODE 1

Mode 1 is the same as Mode 0, except that the Timer register is being run with all 16 bits.

#### MODE 2

Mode 2 configures the timer register as an 8-bit counter (TL1) with automatic reload, as shown in Figure 7-11. Overflow from TL1 not only sets TF1, but also reloads TL1 with the contents of TH1, which is preset by software. The reload leaves TH1 unchanged.

Mode 2 operation is the same for Timer/Counter 0.

#### MODE 3

Timer 1 in Mode 3 simply holds its count. The effect is the same as setting TR1 = 0.

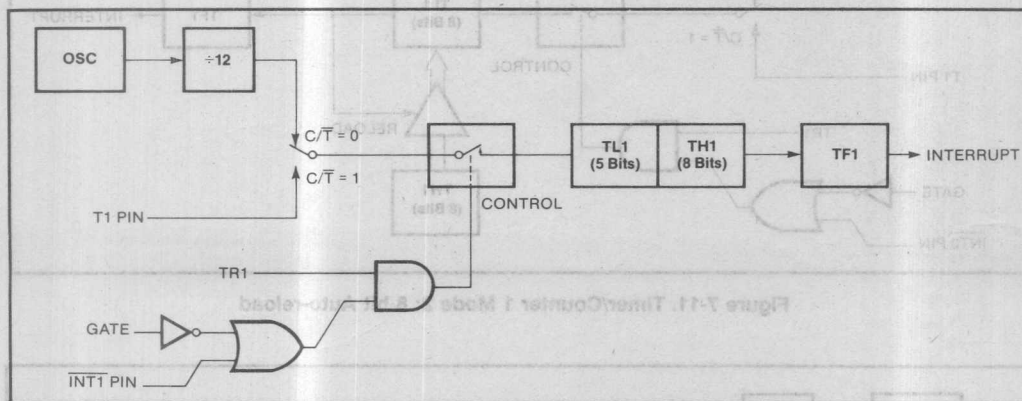


Figure 7-9. Timer/Counter 1 Mode 0: 13-bit Counter

(MSB)				(LSB)			
TF1	TR1	TF0	TR0	IE1	IT1	IE0	IT0
Symbol	Position	Name and Significance		Symbol	Position	Name and Significance	
TF1	TCON.7	Timer 1 overflow Flag. Set by hardware on timer/counter overflow. Cleared by hardware when processor vectors to interrupt routine.		IE1	TCON.3	Interrupt 1 Edge flag. Set by hardware when external interrupt edge detected. Cleared when interrupt processed.	
TR1	TCON.6	Timer 1 Run control bit. Set/cleared by software to turn timer/counter on/off.		IT1	TCON.2	Interrupt 1 Type control bit. Set/cleared by software to specify falling edge/low level triggered external interrupts.	
TF0	TCON.5	Timer 0 overflow Flag. Set by hardware on timer/counter overflow. Cleared by hardware when processor vectors to interrupt routine.		IE0	TCON.1	Interrupt 0 Edge flag. Set by hardware when external interrupt edge detected. Cleared when interrupt processed.	
TR0	TCON.4	Timer 0 Run control bit. Set/cleared by software to turn timer/counter on/off.		IT0	TCON.0	Interrupt 0 Type control bit. Set/cleared by software to specify falling edge/low level triggered external interrupts.	

Figure 7-10. TCON: Timer/Counter Control Register

Timer 0 in Mode 3 establishes TL0 and TH0 as two separate counters. The logic for Mode 3 on Timer 0 is shown in Figure 7-12. TL0 uses the Timer 0 control bits: C/T, GATE, TR0, INT0, and TF0. TH0 is locked into a timer function (counting machine cycles) and takes over the use of TR1 and TF1 from Timer 1. Thus, TH0 now controls the "Timer 1" interrupt.

Mode 3 is provided for applications requiring an extra 8-bit timer or counter. With Timer 0 in Mode 3, an 8051 can look like it has three timer/counters, and an 8052, like it has four. When Timer 0 is in Mode 3, Timer 1 can be

turned on and off by switching it out of and into its own Mode 3, or can still be used by the serial port as a baud rate generator, or in fact, in any application not requiring an interrupt.

## 7.6.2 Timer 2

Timer 2 is a 16-bit timer/counter which is present only in the 8052. Like Timers 0 and 1, it can operate either as a timer or as an event counter. This is selected by bit C/T2 in the Special Function Register T2CON (Figure 7-13). It has three operating modes: "capture," "auto-

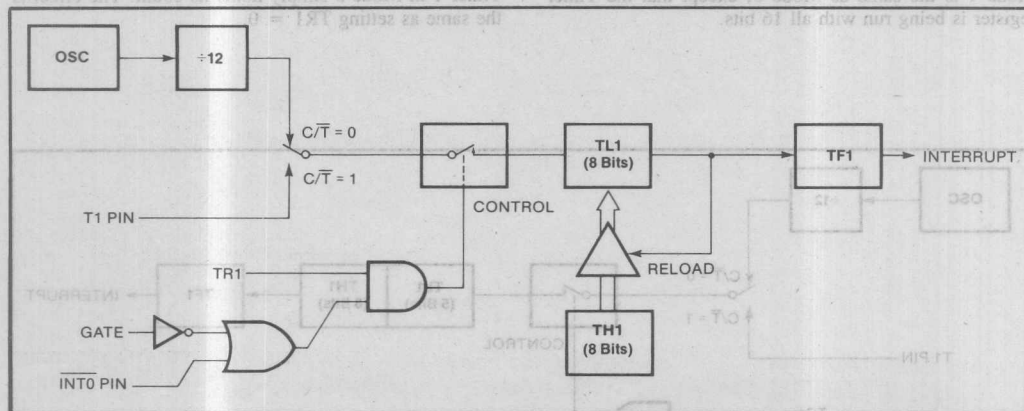


Figure 7-11. Timer/Counter 1 Mode 2: 8-bit Auto-reload

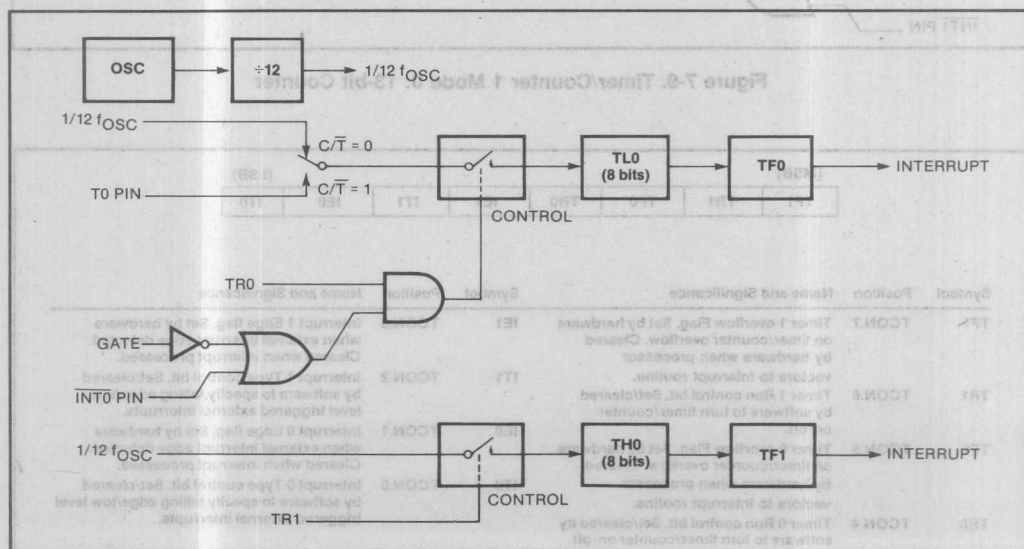


Figure 7-12. Timer/Counter 0 Mode 3: Two 8-bit Counters

(MSB)				(LSB)			
TF2	EXF2	RCLK	TCLK	EXEN2	TR2	C/T2	CP/RL2
Symbol	Position	Name and Significance					
TF2	T2CON.7	Timer 2 overflow flag set by a Timer 2 overflow and must be cleared by software. TF2 will not be set when either RCLK = 1 or TCLK = 1.					
EXF2	T2CON.6	Timer 2 external flag set when either a capture or reload is caused by a negative transition on T2EX and EXEN2 = 1. When Timer 2 interrupt is enabled, EXF2 = 1 will cause the CPU to vector to the Timer 2 Interrupt routine. EXF2 must be cleared by software.					
RCLK	T2CON.5	Receive clock flag. When set, causes the serial port to use Timer 2 overflow pulses for its receive clock in modes 1 and 3. RCLK = 0 causes Timer 1 overflow to be used for the receive clock.					
TCLK	T2CON.4	Transmit clock flag. When set, causes the serial port to use Timer 2 overflow pulses for its transmit clock in modes 1 and 3. TCLK = 0 causes Timer 1 overflows to be used for the transmit clock.					
EXEN2	T2CON.3	Timer 2 external enable flag. When set, allows a capture or reload to occur as a result of a negative transition on T2EX if Timer 2 is not being used to clock the serial port. EXEN2 = 0 causes Timer 2 to ignore events at T2EX.					
TR2	T2CON.2	Start/stop control for Timer 2. A logic 1 starts the timer.					
C/T2	T2CON.1	Timer or counter select. (Timer 2) 0 = Internal timer (OSC/12) 1 = External event counter (falling edge triggered).					
CP/RL2	T2CON.0	Capture/Reload flag. When set, captures will occur on negative transitions at T2EX if EXEN2 = 1. When cleared, auto reloads will occur either with Timer 2 overflows or negative transitions at T2EX when EXEN2 = 1. When either RCLK = 1 or TCLK = 1, this bit is ignored and the timer is forced to auto-reload on Timer 2 overflow.					

Figure 7-13. T2CON: Timer/Counter 2 Control Register

load" and "baud rate generator," which are selected by bits in T2CON as shown in Table 2.

Table 2. Timer 2 Operating Modes

RCLK + TCLK	CP/RL2	TR2	MODE
0	0	1	16-bit auto-reload
0	1	1	16-bit capture
1	X	1	baud rate generator
X	X	0	(off)

In the capture mode there are two options which are selected by bit EXEN2 in T2CON. If EXEN2 = 0, then Timer 2 is a 16-bit timer or counter which upon overflowing sets bit TF2, the Timer 2 overflow bit, which can be used to generate an interrupt. If EXEN2 = 1, then Timer 2 still does the above, but with the added feature that a 1-to-0 transition at external input T2EX causes the current value in the Timer 2 registers, TL2 and TH2, to be captured into registers RCAP2L and RCAP2H, respectively. (RCAP2L and RCAP2H are new Special Function Registers in the 8052.) In addition, the transition at T2EX causes bit EXF2 in T2CON to be set, and EXF2, like TF2, can generate an interrupt.

The capture mode is illustrated in Figure 7-14.

In the auto-reload mode there are again two options, which are selected by bit EXEN2 in T2CON. If EXEN2 = 0, then when Timer 2 rolls over it not only sets TF2 but also causes the Timer 2 registers to be reloaded with the 16-bit value in registers RCAP2L and RCAP2H, which are preset by software. If EXEN2 = 1, then Timer 2 still does the above, but with the added feature that a 1-to-0 transition at external input T2EX will also trigger the 16-bit reload and set EXF2.

The auto-reload mode is illustrated in Figure 7-15.

The baud rate generator mode is selected by RCLK = 1 and/or TCLK = 1. It will be described in conjunction with the serial port.

## 7.7 SERIAL INTERFACE

The serial port is full duplex, meaning it can transmit and receive simultaneously. It is also receive-buffered, meaning it can commence reception of a second byte before a previously received byte has been read from the receive register. (However, if the first byte still hasn't been read by the time reception of the second byte is complete, one of the bytes will be lost). The serial port receive and transmit registers are both accessed at Special Function Register SBUF. Writing to SBUF loads the transmit reg-



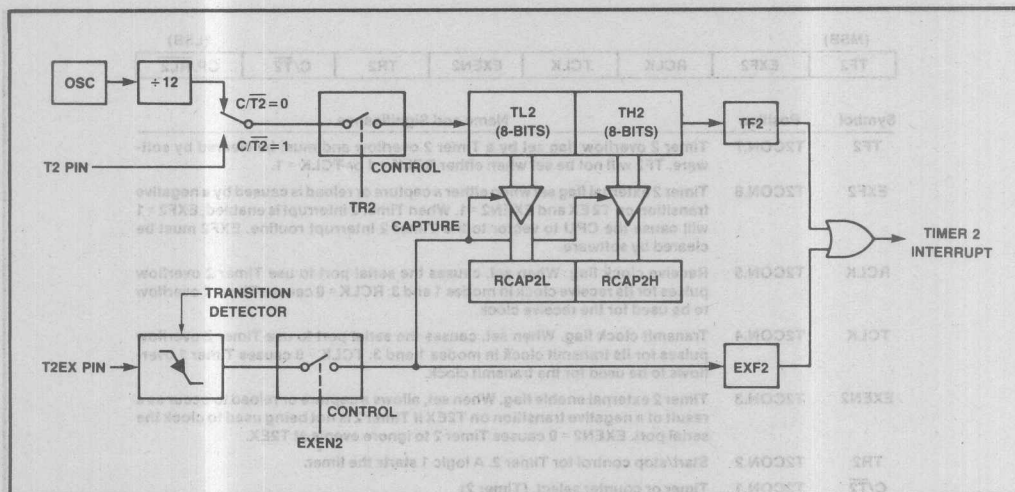


Figure 7-14. Timer 2 in Capture Mode

ister, and reading SBUF accesses a physically separate receive register.

The serial port can operate in 4 modes:

**Mode 0:** Serial data enters and exits through RXD. TXD outputs the shift clock. 8 bits are transmitted/received: 8 data bits (LSB first). The baud rate is fixed at 1/12 the oscillator frequency.

**Mode 1:** 10 bits are transmitted (through TXD) or received (through RXD): a start bit (0), 8 data bits (LSB first), and a stop bit (1). On receive, the stop bit goes into RB8 in Special Function Register SCON. The baud rate is variable.

**Mode 2:** 11 bits are transmitted (through TXD) or received (through RXD): a start bit (0), 8 data bits (LSB first), a programmable 9th data bit, and a stop bit (1). On transmit, the 9th data bit (TB8 in SCON) can be assigned the value of 0 or 1. Or, for example, the parity bit (P, in the PSW) could be moved into TB8. On receive, the 9th data bit goes into RB8 in Special Function Register SCON, while the stop bit is ignored. The baud rate is programmable to either 1/32 or 1/64 the oscillator frequency.

**Mode 3:** 11 bits are transmitted (through TXD) or received (through RXD): a start bit (0), 8 data bits (LSB first), a programmable 9th data bit and a stop bit (1). In fact, Mode 3 is the same as Mode 2 in all respects except the baud rate. The baud rate in Mode 3 is variable.

In all four modes, transmission is initiated by any instruction that uses SBUF as a destination register. Reception is initiated in Mode 0 by the condition RI = 0 and REN = 1. Reception is initiated in the other modes by the incoming start bit if REN = 1.

## 7.7.1 Multiprocessor Communications

Modes 2 and 3 have a special provision for multiprocessor communications. In these modes, 9 data bits are received. The 9th one goes into RB8. Then comes a stop bit. The port can be programmed such that when the stop bit is received, the serial port interrupt will be activated only if RB8 = 1. This feature is enabled by setting bit SM2 in SCON. A way to use this feature in multiprocessor systems is as follows.

When the master processor wants to transmit a block of data to one of several slaves, it first sends out an address byte which identifies the target slave. An address byte differs from a data byte in that the 9th bit is 1 in an address byte and 0 in a data byte. With SM2 = 1, no slave will be interrupted by a data byte. An address byte, however, will interrupt all slaves, so that each slave can examine the received byte and see if it is being addressed. The addressed slave will clear its SM2 bit and prepare to receive the data bytes that will be coming. The slaves that weren't being addressed leave their SM2s set and go on about their business, ignoring the coming data bytes.

SM2 has no effect in Mode 0, and in Mode 1 can be used to check the validity of the stop bit. In a Mode 1 reception, if SM2 = 1, the receive interrupt will not be activated unless a valid stop bit is received.

## 7.7.2 Serial Port Control Register

The serial port control and status register is the Special Function Register SCON, shown in Figure 7-16. This register contains not only the mode selection bits, but also the 9th data bit for transmit and receive (TB8 and RB8), and the serial port interrupt bits (TI and RI).

(MSB)					(LSB)				
SM0	SM1	SM2	REN	TB8	RB8	TI	RI		
where SM0, SM1 specify the serial port mode, as follows:									
SM0	SM1	Mode	Description	Baud Rate					
0	0	0	shift register	$f_{osc}/12$					
0	1	1	8-bit UART	variable					
1	0	2	9-bit UART	$f_{osc}/64$ or $f_{osc}/32$					
1	1	3	9-bit UART variable						
<ul style="list-style-type: none"> <li>SM2 enables the multiprocessor communication feature in modes 2 and 3. In mode 2 or 3, if SM2 is set to 1 then RI will not be activated if the received 9th data bit (RB8) is 0. In mode 1, if SM2 = 1 then RI will not be activated if a valid stop bit was not received. In mode 0, SM2 should be 0.</li> <li>REN enables serial reception. Set by software to enable reception. Clear by software to disable reception.</li> </ul>					<ul style="list-style-type: none"> <li>TB8 is the 9th data bit that will be transmitted in modes 2 and 3. Set or clear by software as desired.</li> <li>RB8 in modes 2 and 3, is the 9th data bit that was received. In mode 1, if SM2 = 0, RB8 is the stop bit that was received. In mode 0, RB8 is not used.</li> <li>TI is transmit interrupt flag. Set by hardware at the end of the 8th bit time in mode 0, or at the beginning of the stop bit in the other modes, in any serial transmission. Must be cleared by software.</li> <li>RI is receive interrupt flag. Set by hardware at the end of the 8th bit time in mode 0, or halfway through the stop bit time in the other modes, in any serial reception (except see SM2). Must be cleared by software.</li> </ul>				

### 7.7.3 Baud Rates

$$\text{Mode 0 Baud Rate} = \frac{\text{Oscillator Frequency}}{12}$$
$$\text{Mode 2 Baud Rate} = \frac{2^{\text{SMOD}}}{64} \times (\text{Oscillator Frequency})$$

In the 8051, the baud rates in Modes 1 and 3 are determined by the Timer 1 overflow rate. In the 8052, these baud rates can be determined by Timer 1, or by Timer 2, or by both (one for transmit and the other for receive).

#### Using Timer 1 to Generate Baud Rates

When Timer 1 is used as the baud rate generator, the baud rates in Modes 1 and 3 are determined by the Timer 1 overflow rate and the value of SMOD as follows:

$$\text{Baud Rate} = \frac{2^{\text{SMOD}}}{32} \times (\text{Timer 1 Overflow Rate})$$

The Timer 1 interrupt should be disabled in this application. The Timer itself can be configured for either "timer" or "counter" operation, and in any of its 3 running modes. In the most typical applications, it is configured for "timer" operation, in the auto-reload mode (high nibble of TMOD = 0010B). In that case, the baud rate is given by the formula

$$\text{Modes 1, 3 Baud Rate} = \frac{2^{\text{SMOD}}}{32} \times \frac{\text{Oscillator Frequency}}{12 \times [256 - (\text{TH1})]}$$

One can achieve very low baud rates with Timer 1 by leaving the Timer 1 interrupt enabled, and configuring the Timer to run as a 16-bit timer (high nibble of TMOD

= 0001B), and using the Timer 1 interrupt to do a 16-bit software reload.

Figure 7-17 lists various commonly used baud rates and how they can be obtained from Timer 1.

BAUD RATE	f <sub>osc</sub>	SMOD	TIMER 1		
			C/T	MODE	RELOAD VALUE
MODE 0 MAX: 1MHZ	12 MHZ	X	X	X	X
MODE 2 MAX: 375K	12 MHZ	1	X	X	X
MODES 1,3: 62.5K	12 MHZ	1	0	2	FFH
19.2K	11.059 MHZ	1	0	2	FDH
9.6K	11.059 MHZ	0	0	2	FDH
4.8K	11.059 MHZ	0	0	2	FAH
2.4K	11.059 MHZ	0	0	2	F4H
1.2K	11.059 MHZ	0	0	2	E8H
137.5	11.986 MHZ	0	0	2	1DH
110	6 MHZ	0	0	2	72H
110	12 MHZ	0	0	1	FE8H

Figure 7-17. Timer 1 Generated Commonly Used Baud Rates

#### Using Timer 2 to Generate Baud Rates

In the 8052, Timer 2 is selected as the baud rate generator by setting TCLK and/or RCLK in T2CON (Figure 7-13). Note then the baud rates for transmit and receive can be simultaneously different. Setting RCLK and/or TCLK puts Timer 2 into its baud rate generator mode, as shown in Figure 7-18.

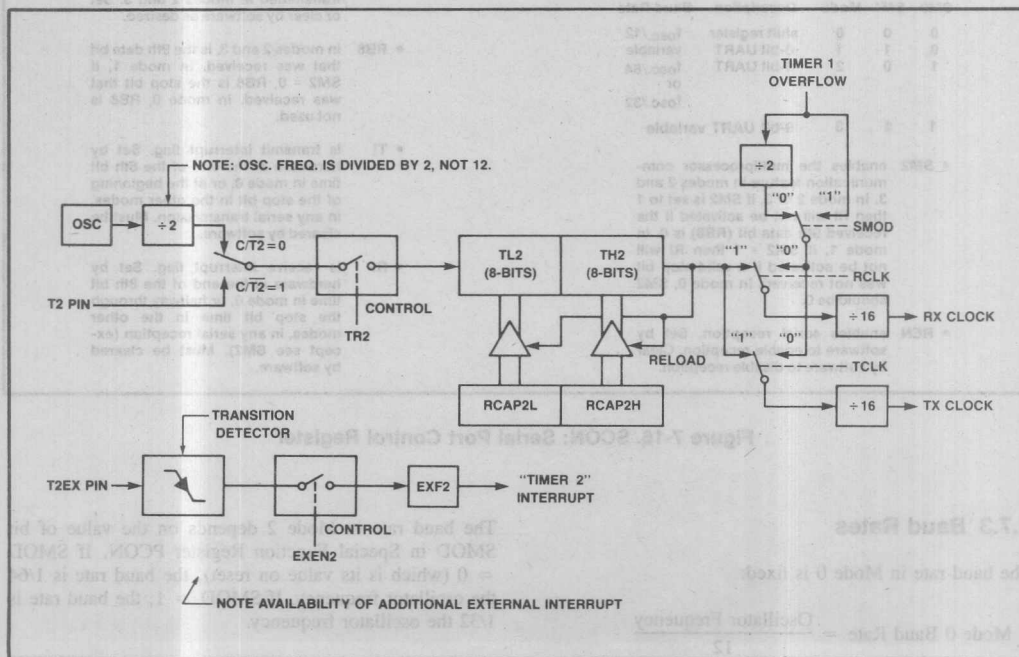


Figure 7-18. Timer 2 in Baud Rate Generator Mode

The baud rate generator mode is similar to the auto-reload mode, in that a rollover in TH2 causes the Timer 2 registers to be reloaded with the 16-bit value in registers RCAP2H and RCAP2L, which are preset by software.

Now, the baud rates in Modes 1 and 3 are determined by Timer 2's overflow rate as follows:

$$\text{Modes 1, 3 Baud Rate} = \frac{\text{Timer 2 Overflow Rate}}{16}$$

The Timer can be configured for either "timer" or "counter" operation. In the most typical applications, it is configured for "timer" operation ( $C/T2 = 0$ ). "Timer" operation is a little different for Timer 2 when it's being used as a baud rate generator. Normally as a timer it would increment every machine cycle (thus at 1/12 the oscillator frequency). As a baud rate generator, however, it increments every state time (thus at 1/2 the oscillator frequency). In that case the baud rate is given by the formula

$$\text{Modes 1, 3 Baud Rate} = \frac{\text{Oscillator Frequency}}{32 \times [65536 - (\text{RCAP2H}, \text{RCAP2L})]}$$

where (RCAP2H, RCAP2L) is the content of RCAP2H and RCAP2L taken as a 16-bit unsigned integer.

Timer 2 as a baud rate generator is shown in Figure 7-18. This Figure is valid only if  $RCLK + TCLK = 1$  in T2CON. Note that a rollover in TH2 does not set TF2, and will not generate an interrupt. Therefore, the Timer 2 interrupt does not have to be disabled when Timer 2 is in the baud rate generator mode. Note too, that if EXEN2 is set, a 1-to-0 transition in T2EX will set EXF2 but will not cause a reload from (RCAP2H, RCAP2L) to (TH2, TL2). Thus when Timer 2 is in use as a baud rate generator, T2EX can be used as an extra external interrupt, if desired.

It should be noted that when Timer 2 is running ( $TR2 = 1$ ) in "timer" function in the baud rate generator mode, one should not try to read or write TH2 or TL2. Under these conditions the Timer is being incremented every state time, and the results of a read or write may not be accurate. The RCAP registers may be read, but shouldn't be written to, because a write might overlap a reload and cause write and/or reload errors. Turn the Timer off (clear TR2) before accessing the Timer 2 or RCAP registers, in this case.

#### 7.7.4 More About Mode 0

Serial data enters and exits through RXD. TXD outputs the shift clock. 8 bits are transmitted/received: 8 data bits (LSB first). The baud rate is fixed at 1/12 the oscillator frequency.

Figure 7-19 shows a simplified functional diagram of the serial port in mode 0, and associated timing.

Transmission is initiated by any instruction that uses SBUF as a destination register. The "write to SBUF" signal at S6P2 also loads a 1 into the 9th bit position of the transmit shift register and tells the TX Control block to commence a transmission. The internal timing is such that one full machine cycle will elapse between "write to SBUF," and activation of SEND.

SEND enables the output of the shift register to the alternate output function line of P3.0, and also enables SHIFT CLOCK to the alternate output function line of P3.1. SHIFT CLOCK is low during S3, S4, and S5 of every machine cycle, and high during S6, S1 and S2. At S6P2 of every machine cycle in which SEND is active, the contents of the transmit shift register are shifted to the right one position.

As data bits shift out to the right, zeros come in from the left. When the MSB of the data byte is at the output position of the shift register, then the 1 that was initially loaded into the 9th position, is just to the left of the MSB, and all positions to the left of that contain zeros. This condition flags the TX Control block to do one last shift and then deactivate SEND and set T1. Both of these actions occur at S1P1 of the 10th machine cycle after "write to SBUF."

Reception is initiated by the condition  $REN = 1$  and  $RI = 0$ . At S6P2 of the next machine cycle, the RX Control unit writes the bits 11111110 to the receive shift register, and in the next clock phase activates RECEIVE.

RECEIVE enables SHIFT CLOCK to the alternate output function line of P3.1. SHIFT CLOCK makes transitions at S3P1 and S6P1 of every machine cycle. At S6P2 of every machine cycle in which RECEIVE is active, the contents of the receive shift register are shifted to the left one position. The value that comes in from the right is the value that was sampled at the P3.0 pin at S5P2 of the same machine cycle.

As data bits come in from the right, 1s shift out to the left. When the 0 that was initially loaded into the rightmost position arrives at the leftmost position in the shift register, it flags the RX Control block to do one last shift and load SBUF. At S1P1 of the 10th machine cycle after the write to SCON that cleared RI, RECEIVE is cleared and RI is set.

#### 7.7.5 More About Mode 1

Ten bits are transmitted (through TXD), or received (through RXD): a start bit (0), 8 data bits (LSB first), and a stop bit (1). On receive, the stop bit goes into RB8 in SCON. In the 8051 the baud rate is determined by the Timer 1 overflow rate. In the 8052 it is determined either by the Timer 1 overflow rate, or the Timer 2 overflow rate, or both (one for transmit and the other for receive).

Figure 7-20 shows a simplified functional diagram of the serial port in Mode 1, and associated timings for transmit and receive.



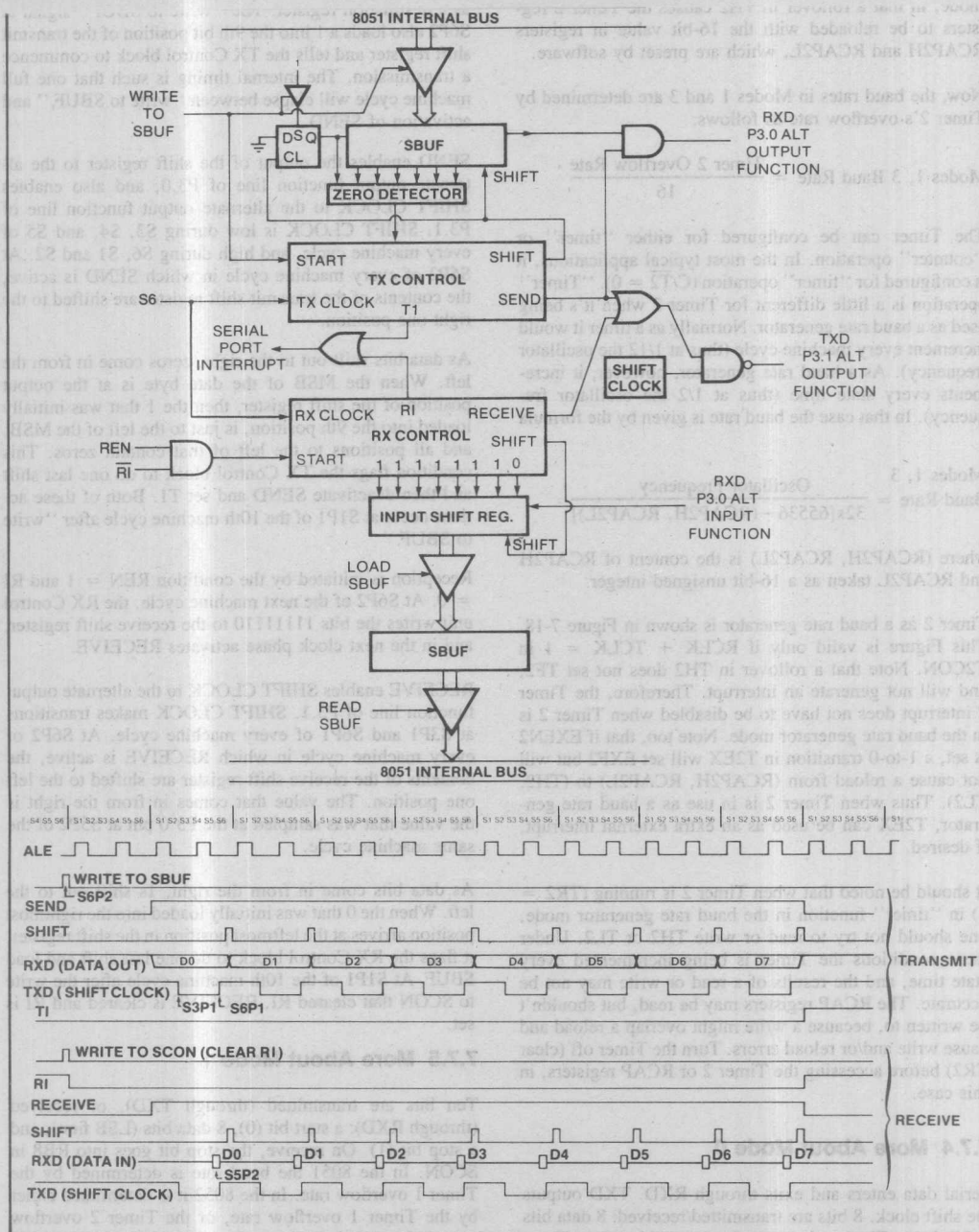
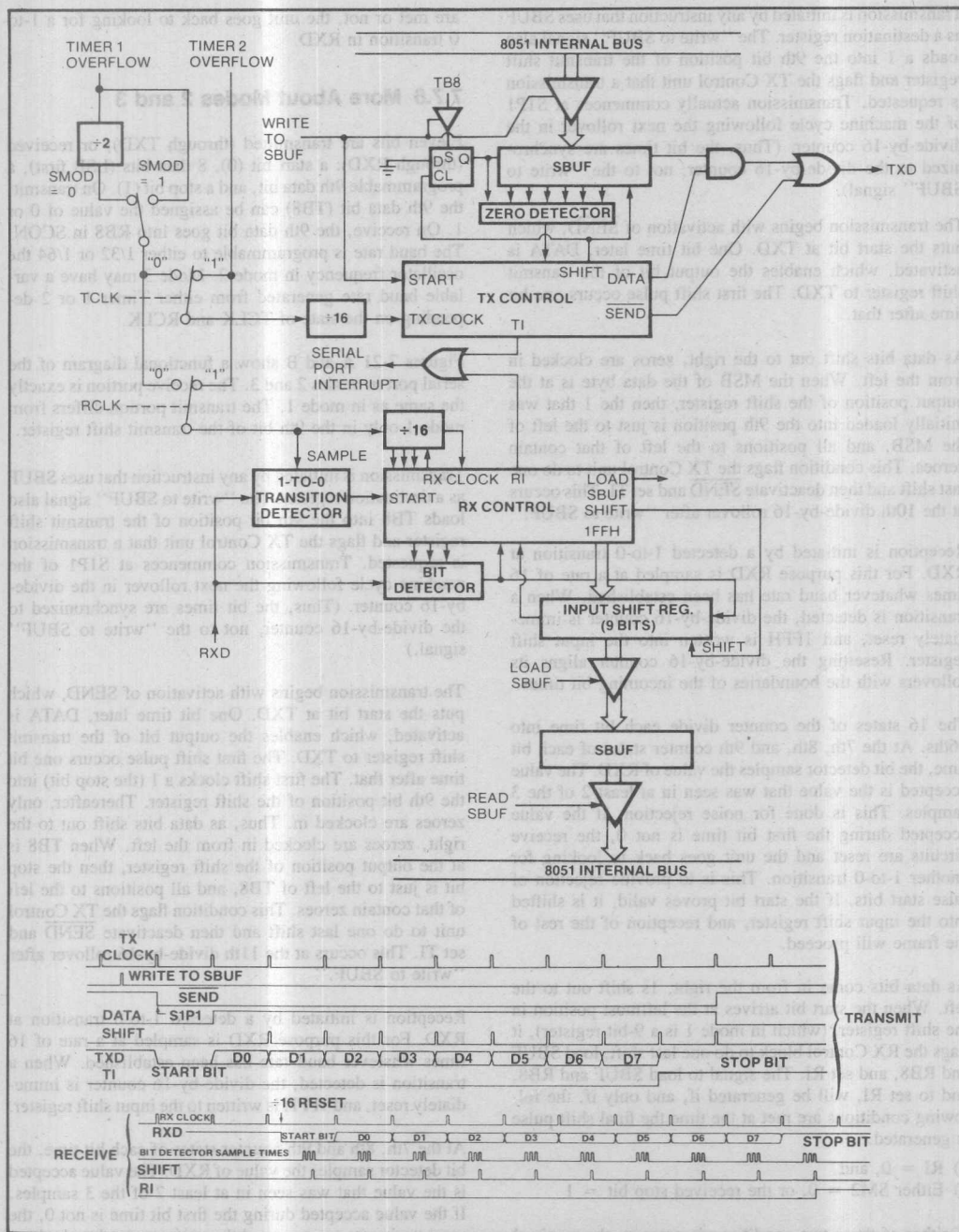


Figure 7-19. Serial Port Mode 0

## MCS®-51 ARCHITECTURE



**Figure 7-20. Serial Port Mode 1**  
TCLK, RCLK, and Timer 2 are present in the 8052/8032 only.

Transmission is initiated by any instruction that uses SBUF as a destination register. The "write to SBUF" signal also loads a 1 into the 9th bit position of the transmit shift register and flags the TX Control unit that a transmission is requested. Transmission actually commences at S1P1 of the machine cycle following the next rollover in the divide-by-16 counter. (Thus, the bit times are synchronized to the divide-by-16 counter, not to the "write to SBUF" signal).

The transmission begins with activation of **SEND**, which puts the start bit at TXD. One bit time later, DATA is activated, which enables the output bit of the transmit shift register to TXD. The first shift pulse occurs one bit time after that.

As data bits shift out to the right, zeros are clocked in from the left. When the MSB of the data byte is at the output position of the shift register, then the 1 that was initially loaded into the 9th position is just to the left of the MSB, and all positions to the left of that contain zeroes. This condition flags the TX Control unit to do one last shift and then deactivate **SEND** and set TI. This occurs at the 10th divide-by-16 rollover after "write to SBUF."

Reception is initiated by a detected 1-to-0 transition at RXD. For this purpose RXD is sampled at a rate of 16 times whatever baud rate has been established. When a transition is detected, the divide-by-16 counter is immediately reset, and 1FFH is written into the input shift register. Resetting the divide-by-16 counter aligns its rollovers with the boundaries of the incoming bit times.

The 16 states of the counter divide each bit time into 16ths. At the 7th, 8th, and 9th counter states of each bit time, the bit detector samples the value of RXD. The value accepted is the value that was seen in at least 2 of the 3 samples. This is done for noise rejection. If the value accepted during the first bit time is not 0, the receive circuits are reset and the unit goes back to looking for another 1-to-0 transition. This is to provide rejection of false start bits. If the start bit proves valid, it is shifted into the input shift register, and reception of the rest of the frame will proceed.

As data bits come in from the right, 1s shift out to the left. When the start bit arrives at the leftmost position in the shift register, (which in mode 1 is a 9-bit register), it flags the RX Control block to do one last shift, load SBUF and RB8, and set RI. The signal to load SBUF and RB8, and to set RI, will be generated if, and only if, the following conditions are met at the time the final shift pulse is generated.

- 1) RI = 0, and
- 2) Either SM2 = 0, or the received stop bit = 1

If either of these two conditions is not met, the received frame is irretrievably lost. If both conditions are met, the stop bit goes into RB8, the 8 data bits go into SBUF, and RI is activated. At this time, whether the above conditions

are met or not, the unit goes back to looking for a 1-to-0 transition in RXD.

## 7.7.6 More About Modes 2 and 3

Eleven bits are transmitted (through TXD), or received (through RXD): a start bit (0), 8 data bits (LSB first), a programmable 9th data bit, and a stop bit (1). On transmit, the 9th data bit (TB8) can be assigned the value of 0 or 1. On receive, the 9th data bit goes into RB8 in SCON. The baud rate is programmable to either 1/32 or 1/64 the oscillator frequency in mode 2. Mode 3 may have a variable baud rate generated from either Timer 1 or 2 depending on the state of TCLK and RCLK.

Figures 7-21 A and B show a functional diagram of the serial port in modes 2 and 3. The receive portion is exactly the same as in mode 1. The transmit portion differs from mode 1 only in the 9th bit of the transmit shift register.

Transmission is initiated by any instruction that uses SBUF as a destination register. The "write to SBUF" signal also loads TB8 into the 9th bit position of the transmit shift register and flags the TX Control unit that a transmission is requested. Transmission commences at S1P1 of the machine cycle following the next rollover in the divide-by-16 counter. (Thus, the bit times are synchronized to the divide-by-16 counter, not to the "write to SBUF" signal.)

The transmission begins with activation of **SEND**, which puts the start bit at TXD. One bit time later, DATA is activated, which enables the output bit of the transmit shift register to TXD. The first shift pulse occurs one bit time after that. The first shift clocks a 1 (the stop bit) into the 9th bit position of the shift register. Thereafter, only zeroes are clocked in. Thus, as data bits shift out to the right, zeroes are clocked in from the left. When TB8 is at the output position of the shift register, then the stop bit is just to the left of TB8, and all positions to the left of that contain zeroes. This condition flags the TX Control unit to do one last shift and then deactivate **SEND** and set TI. This occurs at the 11th divide-by-16 rollover after "write to SBUF."

Reception is initiated by a detected 1-to-0 transition at RXD. For this purpose RXD is sampled at a rate of 16 times whatever baud rate has been established. When a transition is detected, the divide-by-16 counter is immediately reset, and 1FFH is written into the input shift register.

At the 7th, 8th and 9th counter states of each bit time, the bit detector samples the value of RXD. The value accepted is the value that was seen in at least 2 of the 3 samples. If the value accepted during the first bit time is not 0, the receive circuits are reset and the unit goes back to looking for another 1-to-0 transition. If the start bit proves valid, it is shifted into the input shift register, and reception of the rest of the frame will proceed.

# MCS®-51 ARCHITECTURE

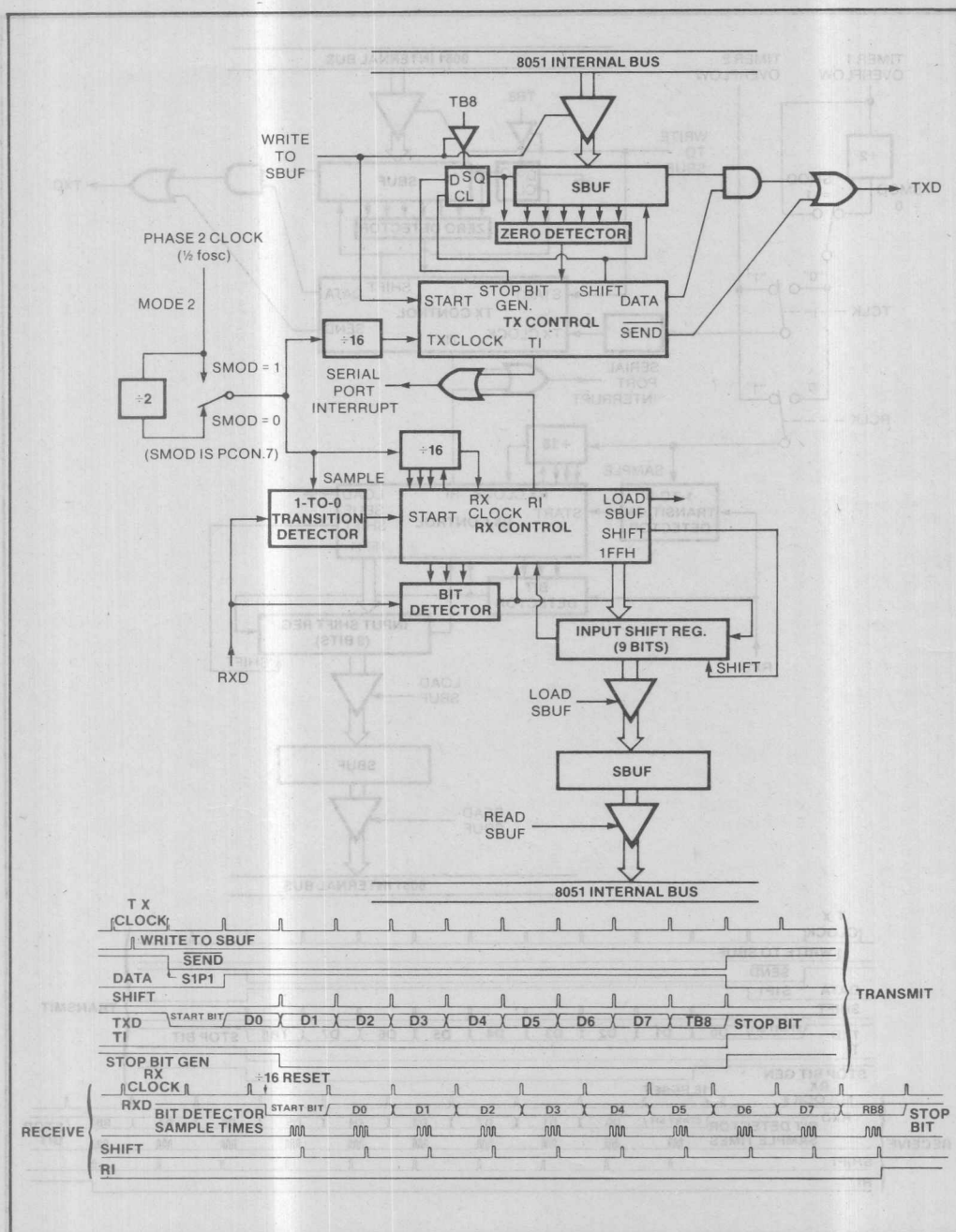


Figure 7-21A. Serial Port Mode 2



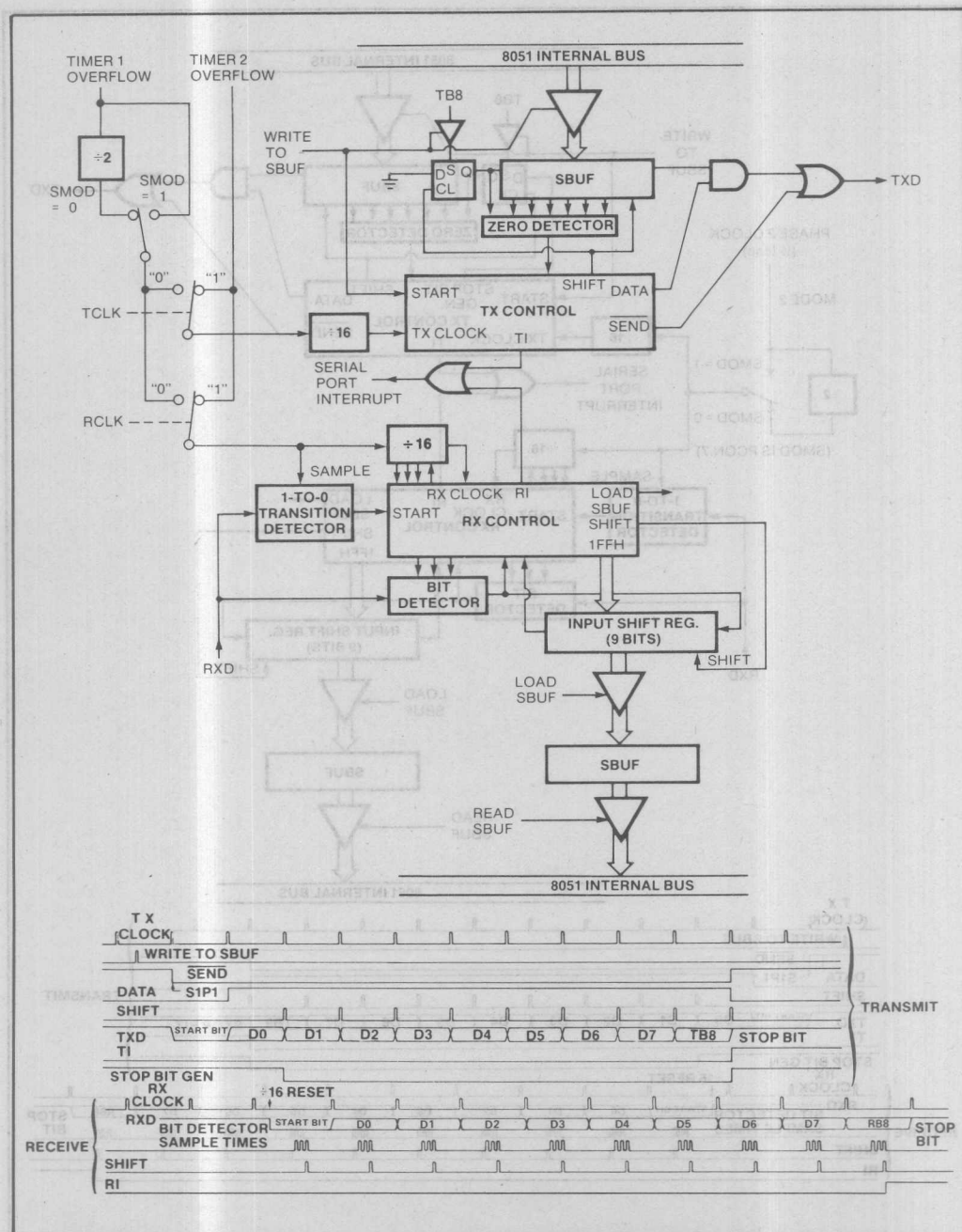


Figure 7-21B. Serial Port Mode 3  
TCLK, RCLK, and Timer 2 are present in the 8052/8032 only.

As data bits come in from the right, 1s shift out to the left. When the start bit arrives at the leftmost position in the shift register (which in modes 2 and 3 is a 9-bit register), it flags the RX Control block to do one last shift, load SBUF and RB8, and set RI. The signal to load SBUF and RB8, and to set RI, will be generated if, and only if, the following conditions are met at the time the final shift pulse is generated:

- 1) RI = 0, and
- 2) Either SM2 = 0 or the received 9th data bit = 1

If either of these conditions is not met, the received frame is irretrievably lost, and RI is not set. If both conditions are met, the received 9th data bit goes into RB8, and the first 8 data bits go into SBUF. One bit time later, whether the above conditions were met or not, the unit goes back to looking for a 1-to-0 transition at the RXD input.

Note that the value of the received stop bit is irrelevant to SBUF, RB8, or RI.

## 7.8 INTERRUPTS

The 8051 provides 5 interrupt sources. The 8052 provides 6. These are shown in Figure 7-22.

The External Interrupts  $\overline{INT0}$  and  $\overline{INT1}$  can each be either level-activated or transition-activated, depending on bits

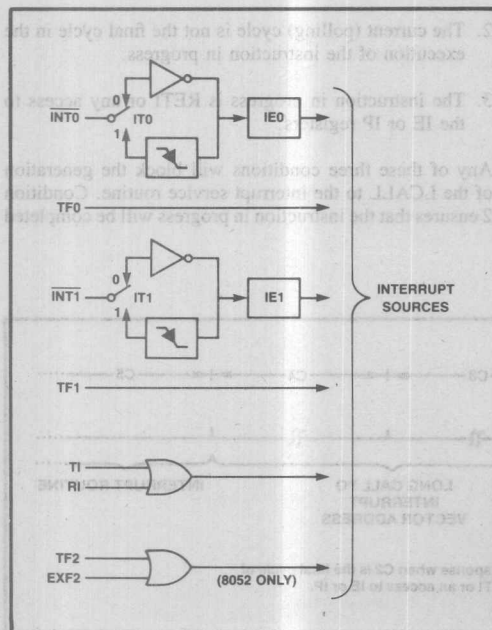


Figure 7-22. MCS-51 Interrupt Sources

IT0 and IT1 in Register TCON. The flags that actually generate these interrupts are bits IE0 and IE1 in TCON. When an external interrupt is generated, the flag that generated it is cleared by the hardware when the service routine is vectored to only if the interrupt was transition-activated. If the interrupt was level-activated, then the external requesting source is what controls the request flag, rather than the on-chip hardware.

The Timer 0 and Timer 1 Interrupts are generated by TF0 and TF1, which are set by a rollover in their respective timer/counter registers (except see Section 7.6.1 for Timer 0 in mode 3). When a timer interrupt is generated, the flag that generated it is cleared by the on-chip hardware when the service routine is vectored to.

The Serial Port Interrupt is generated by the logical OR of RI and TI. Neither of these flags is cleared by hardware when the service routine is vectored to. In fact, the service routine will normally have to determine whether it was RI or TI that generated the interrupt, and the bit will have to be cleared in software.

In the 8052, the Timer 2 Interrupt is generated by the logical OR of TF2 and EXF2. Neither of these flags is cleared by hardware when the service routine is vectored to. In fact, the service routine may have to determine whether it was TF2 or EXF2 that generated the interrupt, and the bit will have to be cleared in software.

All of the bits that generate interrupts can be set or cleared by software, with the same result as though it had been set or cleared by hardware. That is, interrupts can be generated or pending interrupts can be canceled in software.

(MSB)				(LSB)			
EA	X	ET2	ES	ET1	EX1	ET0	EX0
Symbol	Position	Function					
EA	IE.7	disables all interrupts. If EA = 0, no interrupt will be acknowledged. If EA = 1, each interrupt source is individually enabled or disabled by setting or clearing its enable bit.					
—	IE.6	reserved					
ET2	IE.5	enables or disables the Timer 2 overflow or capture interrupt. If ET2 = 0, the Timer 2 interrupt is disabled.					
ES	IE.4	enables or disables the Serial Port interrupt. If ES = 0, the Serial Port interrupt is disabled.					
ET1	IE.3	enables or disables the Timer 1 Overflow interrupt. If ET1 = 0, the Timer 1 interrupt is disabled.					
EX1	IE.2	enables or disables External Interrupt 1. If EX1 = 0, External Interrupt 1 is disabled.					
ET0	IE.1	enables or disables the Timer 0 Overflow interrupt. If ET0 = 0, the Timer 0 interrupt is disabled.					
EX0	IE.0	enables or disables External Interrupt 0. If EX0 = 0, External Interrupt 0 is disabled.					

Figure 7-23. IE: Interrupt Enable Register

Each of these interrupt sources can be individually enabled or disabled by setting or clearing a bit in Special Function Register IE (Figure 7-23). Note that IE contains also a global disable bit, EA, which disables all interrupts at once.

### 7.8.1 Priority Level Structure

Each interrupt source can also be individually programmed to one of two priority levels by setting or clearing a bit in Special Function Register IP (Figure 7-24). A low-priority interrupt can itself be interrupted by a high-priority interrupt, but not by another low-priority interrupt. A high-priority interrupt can't be interrupted by any other interrupt source.

(MSB)		(LSB)	
		X	X
Symbol	Position	Function	
—	IP.7	reserved	
—	IP.6	reserved	
PT2	IP.5	defines the Timer 2 interrupt priority level. PT2 = 1 programs it to the higher priority level.	
PS	IP.4	defines the Serial Port Interrupt priority level. PS = 1 programs it to the higher priority level.	
PT1	IP.3	defines the Timer 1 interrupt priority level. PT1 = 1 programs it to the higher priority level.	
PX1	IP.2	defines the External Interrupt 1 priority level. PX1 = 1 programs it to the higher priority level.	
PT0	IP.1	defines the Timer 0 interrupt priority level. PT0 = 1 programs it to the higher priority level.	
PX0	IP.0	defines the External Interrupt 0 priority level. PX0 = 1 programs it to the higher priority level.	

Figure 7-24. IP: Interrupt Priority Register

If two requests of different priority levels are received simultaneously, the request of higher priority level is serviced. If requests of the **same** priority level are received simultaneously, an internal polling sequence determines which request is serviced. Thus within each priority level there is a second priority structure determined by the polling sequence, as follows:

SOURCE	PRIORITY WITHIN LEVEL
1. IE0	(highest)
2. TF0	
3. IE1	
4. TF1	
5. RI + TI	
6. TF2 + EXF2	(lowest)

Note that the "priority within level" structure is only used to resolve *simultaneous requests of the same priority level*.

### 7.8.2 How Interrupts Are Handled

The interrupt flags are sampled at S5P2 of every machine cycle. The samples are polled during the following machine cycle. If one of the flags was in a set condition at S5P2 of the preceding cycle, the polling cycle will find it and the interrupt system will generate an LCALL to the appropriate service routine, provided this hardware-generated LCALL is not blocked by any of the following conditions:

1. An interrupt of equal or higher priority level is already in progress.
2. The current (polling) cycle is not the final cycle in the execution of the instruction in progress.
3. The instruction in progress is RETI or any access to the IE or IP registers.

Any of these three conditions will block the generation of the LCALL to the interrupt service routine. Condition 2 ensures that the instruction in progress will be completed

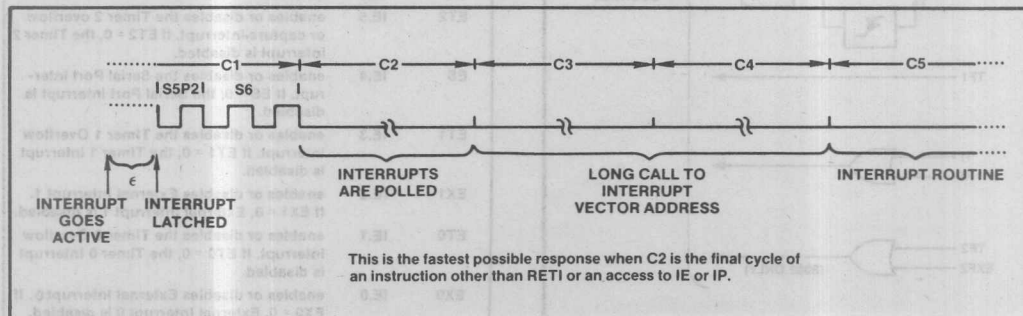


Figure 7-25. Interrupt Response Timing Diagram

before vectoring to any service routine. Condition 3 ensures that if the instruction in progress is RETI or any access to IE or IP, then at least *one more* instruction will be executed before any interrupt is vectored to.

The polling cycle is repeated with each machine cycle, and the values polled are the values that were present at S5P2 of the previous machine cycle. Note then that if an interrupt flag is active but not being responded to for one of the above conditions, if the flag is not *still* active when the blocking condition is removed, the denied interrupt will not be serviced. In other words, the fact that the interrupt flag was once active but not serviced is not remembered. Every polling cycle is new.

The polling cycle/LCALL sequence is illustrated in Figure 7-25.

Note that if an interrupt of higher priority level goes active prior to S5P2 of the machine cycle labeled C3 in Figure 7-25, then in accordance with the above rules it will be vectored to during C5 and C6, without any instruction of the lower priority routine having been executed.

Thus the processor acknowledges an interrupt request by executing a hardware-generated LCALL to the appropriate servicing routine. In some cases it also clears the flag that generated the interrupt, and in other cases it doesn't. It never clears the Serial Port or Timer 2 flags. This has to be done in the user's software. It clears an external interrupt flag (IE0 or IE1) only if it was transition-activated. The hardware-generated LCALL pushes the contents of the Program Counter onto the stack (but it does not save the PSW) and reloads the PC with an address that depends on the source of the interrupt being vectored to, as shown below.

SOURCE	VECTOR ADDRESS
IE0	0003H
TF0	000BH
IE1	0013H
TF1	001BH
RI + TI	0023H
TF2 + EXF2	002BH

Execution proceeds from that location until the RETI instruction is encountered. The RETI instruction informs the processor that this interrupt routine is no longer in progress, then pops the top two bytes from the stack and reloads the Program Counter. Execution of the interrupted program continues from where it left off.

Note that a simple RET instruction would also have returned execution to the interrupted program, but it would have left the interrupt control system thinking an interrupt was still in progress.

## 7.8.3 External Interrupts

The external sources can be programmed to be level-activated or transition-activated by setting or clearing bit IT1 or IT0 in Register TCON. If ITx = 0, external interrupt x is triggered by a detected low at the INTx pin. If ITx = 1, external interrupt x is edge-triggered. In this mode if successive samples of the INTx pin show a high in one cycle and a low in the next cycle, interrupt request flag IEx in TCON is set. Flag bit IEx then requests the interrupt.

Since the external interrupt pins are sampled once each machine cycle, an input high or low should hold for at least 12 oscillator periods to ensure sampling. If the external interrupt is transition-activated, the external source has to hold the request pin high for at least one cycle, and then hold it low for at least one cycle to ensure that the transition is seen so that interrupt request flag IEx will be set. IEx will be automatically cleared by the CPU when the service routine is called.

If the external interrupt is level-activated, the external source has to hold the request active until the requested interrupt is actually generated. Then it has to deactivate the request before the interrupt service routine is completed, or else another interrupt will be generated.

## 7.8.4 Response Time

The INT0 and INT1 levels are inverted and latched into IE0 and IE1 at S5P2 of every machine cycle. The values are not actually polled by the circuitry until the next machine cycle. If a request is active and conditions are right for it to be acknowledged, a hardware subroutine call to the requested service routine will be the next instruction to be executed. The call itself takes two cycles. Thus, a minimum of three complete machine cycles elapse between activation of an external interrupt request and the beginning of execution of the first instruction of the service routine. Figure 7-25 shows interrupt response timings.

A longer response time would result if the request is blocked by one of the 3 previously listed conditions. If an interrupt of equal or higher priority level is already in progress, the additional wait time obviously depends on the nature of the other interrupt's service routine. If the instruction in progress is not in its final cycle, the additional wait time cannot be more than 3 cycles, since the longest instructions (MUL and DIV) are only 4 cycles long, and if the instruction in progress is RETI or an access to IE or IP, the additional wait time cannot be more than 5 cycles (a maximum of one more cycle to complete the instruction in progress, plus 4 cycles to complete the next instruction if the instruction is MUL or DIV).

Thus, in a single-interrupt system, the response time is always more than 3 cycles and less than 8 cycles.



## 7.9 SINGLE-STEP OPERATION

The 8051 interrupt structure allows single-step execution with very little software overhead. As previously noted, an interrupt request will not be responded to while an interrupt of equal priority level is still in progress, nor will it be responded to after RETI until at least one other instruction has been executed. Thus, once an interrupt routine has been entered, it cannot be re-entered until at least once instruction of the interrupted program is executed. One way to use this feature for single-step operation is to program one of the external interrupts (say, INT0) to be level-activated. The service routine for the interrupt will terminate with the following code:

```
JNB P3.2,$ ;WAIT HERE TILL INTO
GOES HIGH
JB P3.2,$ ;NOW WAIT HERE TILL
IT GOES LOW
RETI ;GO BACK AND
EXECUTE ONE
INSTRUCTION
```

Now if the INT0 pin, which is also the P3.2 pin, is held normally low, the CPU will go right into the External Interrupt 0 routine and stay there until INT0 is pulsed (from low to high to low). Then it will execute RETI, go back to the task program, execute one instruction, and immediately re-enter the External Interrupt 0 routine to await the next pulsing of P3.2. One step of the task program is executed each time P3.2 is pulsed.

## 7.10 RESET

The reset input is the RST pin, which is the input to a Schmitt Trigger.

A reset is accomplished by holding the RST pin high for at least two machine cycles (24 oscillator periods), while the oscillator is running. The CPU responds by executing

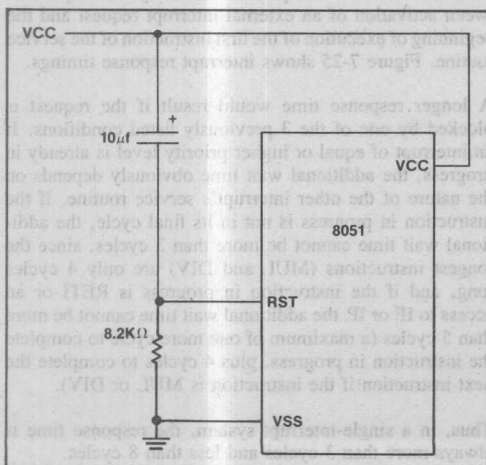


Figure 7-26. Power on Reset Circuit

an internal reset. It also configures the ALE and PSEN pins as inputs. (They are quasi-bidirectional). The internal reset is executed during the second cycle in which RST is high and is repeated every cycle until RST goes low. It leaves the internal registers as follows:

REGISTER	CONTENT
PC	0000H
ACC	00H
B	00H
PSW	00H
SP	07H
DPTR	0000H
P0-P3	0FFH
IP (8051)	XXXX0000B
IP (8052)	XX000000B
IE (8051)	0XX00000B
IE (8052)	0X000000B
TMOD	00H
TCON	00H
T2CON (8052 only)	00H
TH0	00H
TL0	00H
TH1	00H
TL1	00H
TH2	00H
TL2	00H
RCAP2H (8052 only)	00H
RCAP2L (8052 only)	00H
SCON	00H
SBUF	Indeterminate
PCON (HMOS)	0XXXXXXXB
PCON (CHMOS)	0XXX0000B

The internal RAM is not affected by reset. When VCC is turned on, the RAM content is indeterminate unless the part is returning from a reduced power mode of operation.

## POWER-ON RESET

An automatic reset can be obtained when VCC is turned on by connecting the RST pin to VCC through a 10 µF capacitor and to VSS through an 8.2KΩ resistor, providing the VCC risetime does not exceed a millisecond and the oscillator start-up time does not exceed 10 milliseconds. This power-on reset circuit is shown in Figure 7-26. When power comes on, the current drawn by RST commences to charge the capacitor. The voltage at RST is the difference between VCC and the capacitor voltage, and decreases from VCC as the cap charges. The larger the capacitor, the more slowly VRST decreases. VRST must remain above the lower threshold of the Schmitt Trigger long enough to effect a complete reset. The time required is the oscillator start-up time, plus 2 machine cycles.

## 7.11 POWER-SAVING MODES OF OPERATION

For applications where power consumption is a critical factor, both the HMOS and CHMOS versions provide reduced power modes of operation. For the CHMOS ver-

sion of the 8051 the reduced power modes, Idle and Power Down, are standard features. In the HMOS versions a reduced power mode is available, but not as a standard feature. The local sales office will provide ordering information for users requiring this feature.

### 7.11.1 HMOS Power Down Mode

The power down mode in the HMOS devices allows one to reduce VCC to zero while saving the on-chip RAM through a backup supply connected to the RST pin. To use the feature, the user's system, upon detecting that a power failure is imminent, would interrupt the processor in some manner to transfer relevant data to the on-chip RAM and enable the backup power supply to the RST pin before VCC falls below its operating limit. When power returns, the backup supply needs to stay on long enough to accomplish a reset, and then can be removed so that normal operation can be resumed.

### 7.11.2 CHMOS Power Reduction Modes

CHMOS versions have two power-reducing modes, Idle and Power Down. The input through which backup power is supplied during these operations is VCC. Figure 7-27 shows the internal circuitry which implements these features. In the Idle mode (IDL = 1), the oscillator continues to run and the Interrupt, Serial Port, and Timer blocks continue to be clocked, but the clock signal is gated off to the CPU. In Power Down (PD = 1), the oscillator is frozen. The Idle and Power Down modes are activated by setting bits in Special Function Register PCON. The address of this register is 87H. Figure 7-28 details its contents.

#### IDLE MODE

An instruction that sets PCON.0 causes that to be the last instruction executed before going into the Idle mode. In

(MSB)				(LSB)			
SMOD	—	—	—	GF1	GF0	PD	IDL
Symbol	Position	Name and Function					
SMOD	PCON.7	Double Baud rate bit. When set to a 1, the baud rate is doubled when the serial port is being used in either modes 1, 2 or 3.					
—	PCON.6	(Reserved)					
—	PCON.5	(Reserved)					
—	PCON.4	(Reserved)					
GF1	PCON.3	General-purpose flag bit.					
GF0	PCON.2	General-purpose flag bit.					
PD	PCON.1	Power Down bit. Setting this bit activates power down operation.					
IDL	PCON.0	Idle mode bit. Setting this bit activates idle mode operation.					
If 1s are written to PD and IDL at the same time, PD takes precedence. The reset value of PCON is (0XX0000).							

Figure 7-28. PCON: Power Control Register

the Idle mode, the internal clock signal is gated off to the CPU, but not to the Interrupt, Timer, and Serial Port functions. The CPU status is preserved in its entirety: the Stack Pointer, Program Counter, Program Status Word, Accumulator, and all other registers maintain their data during Idle. The port pins hold the logical states they had at the time Idle was activated. ALE and PSEN hold at logic high levels.

There are two ways to terminate the Idle. Activation of any enabled interrupt will cause PCON.0 to be cleared by hardware, terminating the Idle mode. The interrupt will be serviced, and following RETI the next instruction to be executed will be the one following the instruction that put the device into Idle.

The flag bits GF0 and GF1 can be used to give an indication if an interrupt occurred during normal operation or

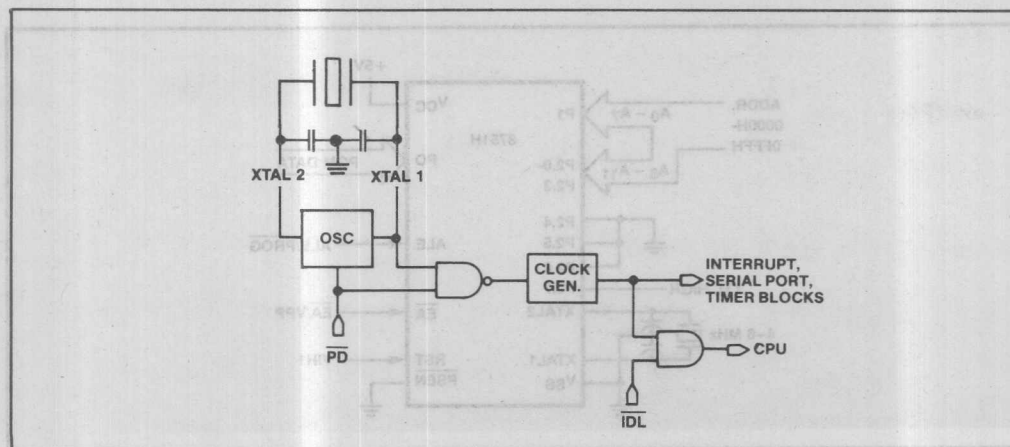


Figure 7-27. Idle and Power Down Hardware

during an Idle. For example, an instruction that activates Idle can also set one or both flag bits. When Idle is terminated by an interrupt, the interrupt service routine can examine the flag bits.

The other way of terminating the Idle mode is with a hardware reset. Since the clock oscillator is still running, the hardware reset needs to be held active for only two machine cycles (24 oscillator periods) to complete the reset.

### POWER DOWN MODE

An instruction that sets PCON.1 causes that to be the last instruction executed before going into the Power Down mode. In the Power Down mode, the on-chip oscillator is stopped. With the clock frozen, all functions are stopped, but the on-chip RAM and Special Function Registers are held. The port pins output the values held by their respective SFRs. ALE and PSEN output lows.

The only exit from Power Down is a hardware reset. Reset redefines all the SFRs, but does not change the on-chip RAM.

In the Power down mode of operation, VCC can be reduced to minimize power consumption. Care must be taken, however, to ensure that VCC is not reduced before the Power Down mode is invoked, and that VCC is restored to its normal operating level, before the Power Down mode is terminated. The reset that terminates Power Down also frees the oscillator. The reset should not be activated before VCC is restored to its normal operating level, and must be held active long enough to allow the oscillator to restart and stabilize (normally less than 10 msec).

### 7.12 8751H

The 8751H is the EPROM member of the MCS-51 family. This means that the on-chip Program Memory can be electrically programmed, and can be erased by exposure to ultraviolet light. The 8751H also has a provision for denying external access to the on-chip Program Memory, in order to protect its contents against software piracy.

#### 7.12.1 Programming the EPROM

To be programmed, the 8751H must be running with a 4 to 6 MHz oscillator. (The reason the oscillator needs to be running is that the internal bus is being used to transfer address and program data to appropriate internal registers.) The address of an EPROM location to be programmed is applied to Port 1 and pins P2.0–P2.3 of Port 2, while the data byte is applied to Port 0. Pins P2.4–P2.6 and PSEN should be held low, and P2.7 and RST high. (These are all TTL levels except RST, which requires 2.5V for a logic high.)  $\overline{EA}/VPP$  is held normally high, and is pulsed to +21V. While  $\overline{EA}/VPP$  is at 21V, the ALE/PROG pin, which is normally being held high, is pulsed low for 50 msec. Then  $\overline{EA}/VPP$  is returned to high. This setup is shown in Figure 7.29. Detailed timing specifications are provided in the 8751H data sheet.

Note: The  $\overline{EA}$  pin must not be allowed to go above the maximum specified VPP level of 21.5V for any amount of time. Even a narrow glitch above that voltage level can cause permanent damage to the device. The VPP source should be well regulated and free of glitches.

#### 7.12.2 Program Verification

If the program security bit has not been programmed, the on-chip Program Memory can be read out for verification

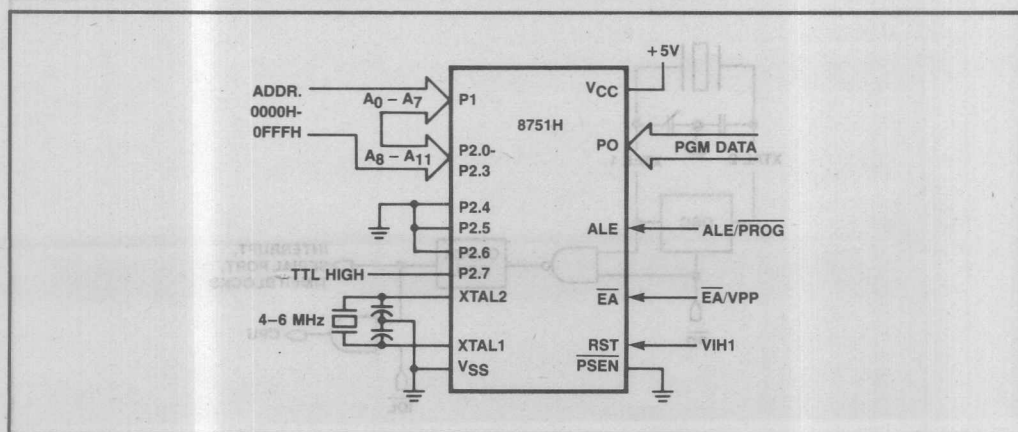


Figure 7-29. Programming the 8751H

purposes, if desired, either during or after the programming operation. The required setup, which is shown in Figure 7.30, is the same as for programming the EPROM except that pin P2.7 is held at TTL low (or used as an active-low read strobe). The address of the Program Memory location to be read is applied to Port 1 and pins P2.0–P2.3. The other Port 2 pins and PSEN are held low. ALE,  $\overline{\text{EA}}$ , and RST are held high. The contents of the addressed location will come out on Port 0. External pull-ups are required on Port 0 for this operation.

### 7.12.3 Program Memory Security

The 8751H contains a security bit, which, once programmed, denies electrical access by any external means to the on-chip Program Memory. The setup and procedure for programming the security bit are the same as for normal

programming, except that pin P2.6 is held at TTL high. The setup is shown in Figure 7.31. Port 0, Port 1, and pins P2.0–P2.3 of Port 2 may be in any state.

Once the security bit has been programmed, it can be deactivated only by full erasure of the Program Memory. While it is programmed, the internal Program Memory cannot be read out, the device cannot be further programmed, and it *cannot execute external program memory*. Erasing the EPROM, thus deactivating the security bit, restores the device's full functionality. It can then be re-programmed.

### 7.12.4 Erasure Characteristics

Erasure of the 8751H Program Memory begins to occur when the chip is exposed to light with wavelengths shorter

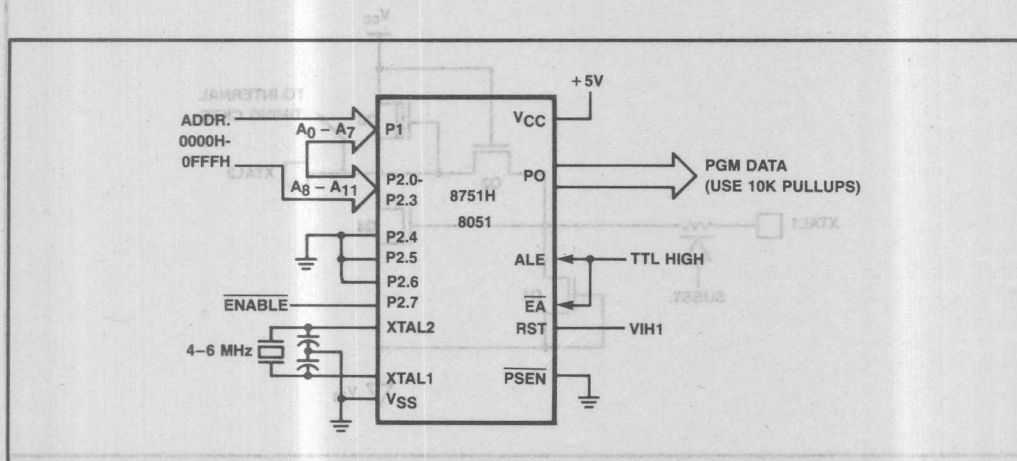


Figure 7-30. Program Verification in the 8751H and 8051

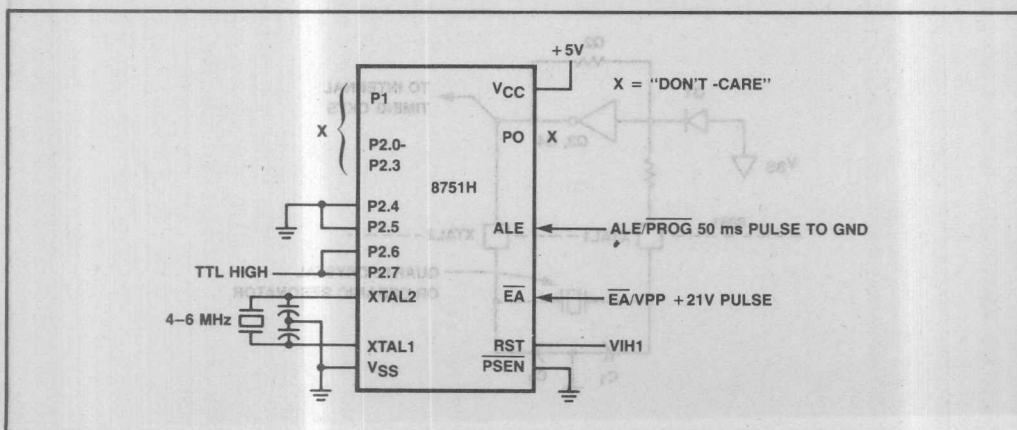


Figure 7-31. Programming the Security Bit in the 8751H



than approximately 4,000 Angstroms. Since sunlight and fluorescent lighting have wavelengths in this range, exposure to these light sources over an extended time (about 1 week in sunlight, or 3 years in room-level fluorescent lighting) could cause inadvertent erasure. If an application subjects the 8751H to this type of exposure, it is suggested that an opaque label be placed over the window.

The recommended erasure procedure is exposure to ultraviolet light (at 2537 Angstroms) to an integrated dose of at least 15 W/cm<sup>2</sup>. Exposing the 8751H to an ultraviolet lamp of 12,000  $\mu$ W/cm<sup>2</sup> rating for 20 to 30 minutes, at a distance of about 1 inch, should be sufficient.

Erasure leaves the array in an all 1s state.

## 7.13 MORE ABOUT THE ON-CHIP OSCILLATOR

### 7.13.1 HMOS Versions

The on-chip oscillator circuitry for the HMOS<sup>®</sup> (HMOS-I and HMOS-II) members of the MCS-51 family is a single stage linear inverter (Figure 7-32), intended for use as a crystal-controlled, positive reactance oscillator (Figure 7-33). In this application the crystal is operated in its fundamental response mode as an inductive reactance in parallel resonance with capacitance external to the crystal.

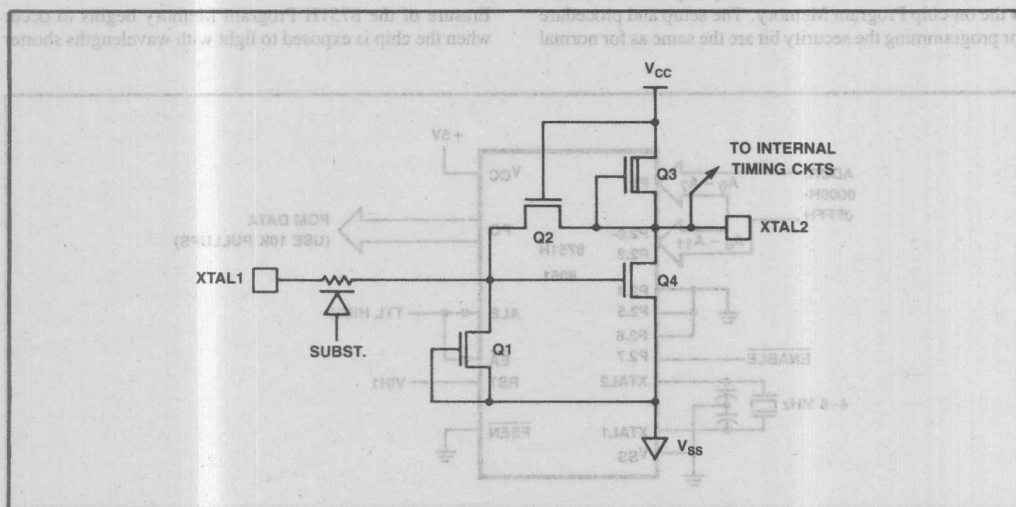


Figure 7-32. On-Chip Oscillator Circuitry in the HMOS Versions of the MCS-51 Family

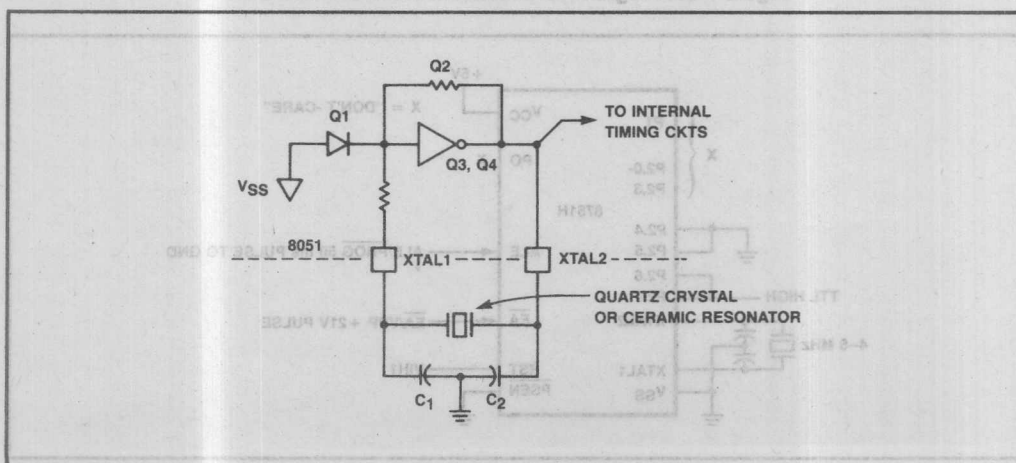
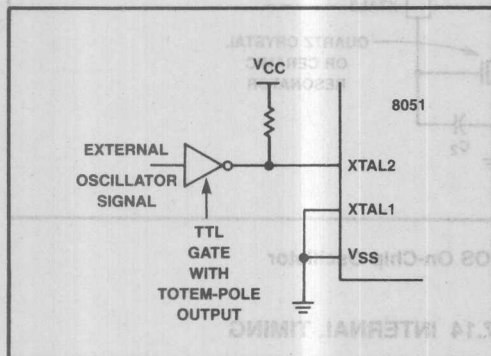


Figure 7-33. Using the HMOS On-Chip Oscillator

The crystal specifications and capacitance values (C1 and C2 in Figure 7-33) are not critical. 30 pF can be used in these positions at any frequency with good quality crystals. A ceramic resonator can be used in place of the crystal in cost-sensitive applications. When a ceramic resonator is used, C1 and C2 are normally selected to be of somewhat higher values, typically, 47 pF. The manufacturer of the ceramic resonator should be consulted for recommendations on the values of these capacitors.

A more in-depth discussion of crystal specifications, ce-



**Figure 7-34. Driving the HMOS MCS-51 Parts with an External Clock Source**

ramic resonators, and the selection of values for C1 and C2 can be found in Application Note AP-155, "Oscillators for Microcontrollers," which is included in this manual.

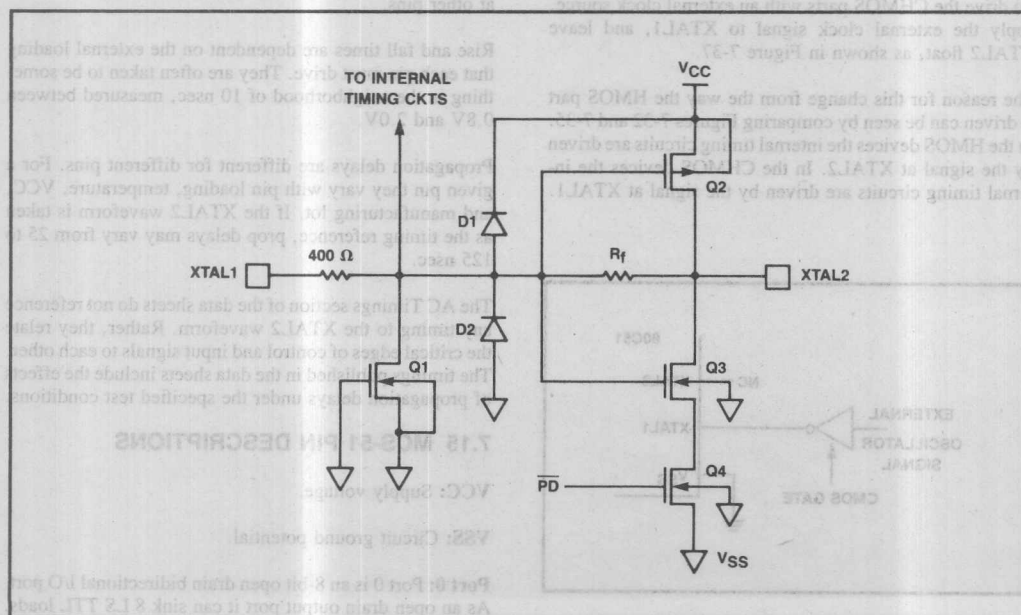
To drive the HMOS parts with an external clock source, apply the external clock signal to XTAL2, and ground XTAL1, as shown in Figure 7-34. A pull-up resistor may be used (to increase noise margin), but is optional if V<sub>OH</sub> of the driving gate exceeds the VIHMIN specification of XTAL2.

### 7.13.2 CHMOS

The on-chip oscillator circuitry for the 80C51, shown in Figure 7-35, consists of a single stage linear inverter intended for use as a crystal-controlled, positive reactance oscillator in the same manner as the HMOS parts. However, there are some important differences.

One difference is that the 80C51 is able to turn off its oscillator under software control (by writing a 1 to the PD bit in PCON). Another difference is that in the 80C51 the internal clocking circuitry is driven by the signal at XTAL1, whereas in the HMOS versions it is by the signal at XTAL2.

The feedback resistor R<sub>f</sub> in Figure 7-35 consists of paralleled n- and p-channel FETs controlled by the PD bit, such that R<sub>f</sub> is opened when PD = 1. The diodes D1 and D2, which act as clamps to V<sub>CC</sub> and V<sub>SS</sub>, are parasitic to the R<sub>f</sub> FETs.



**Figure 7-35. On-Chip Oscillator Circuitry in the CHMOS Versions of the MCS-51 Family**

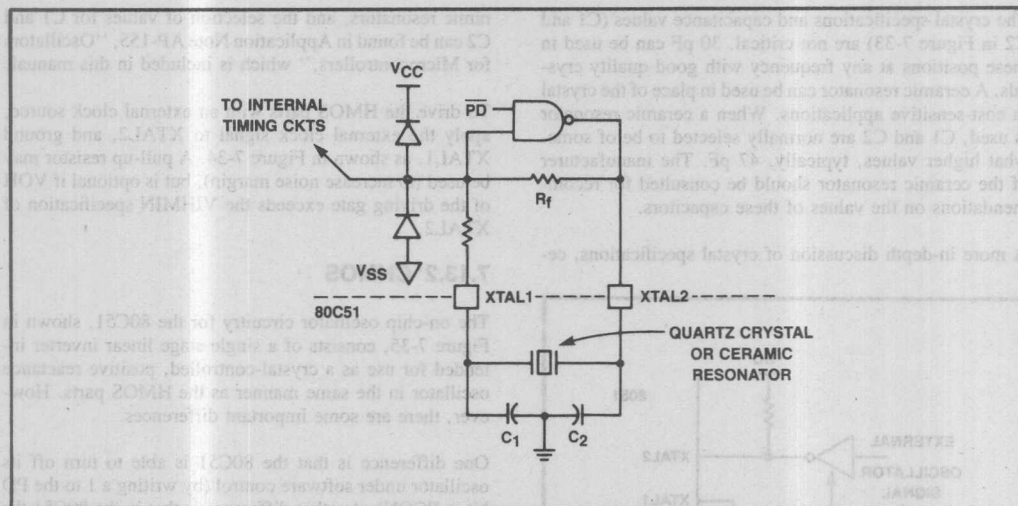


Figure 7-36. Using the CHMOS On-Chip Oscillator

The oscillator can be used with the same external components as the HMOS versions, as shown in Figure 7-36. Typically,  $C1 = C2 = 30 \text{ pF}$  when the feedback element is a quartz crystal, and  $C1 = C2 = 47 \text{ pF}$  when a ceramic resonator is used.

To drive the CHMOS parts with an external clock source, apply the external clock signal to XTAL1, and leave XTAL2 float, as shown in Figure 7-37.

The reason for this change from the way the HMOS part is driven can be seen by comparing Figures 7-32 and 7-35. In the HMOS devices the internal timing circuits are driven by the signal at XTAL2. In the CHMOS devices the internal timing circuits are driven by the signal at XTAL1.

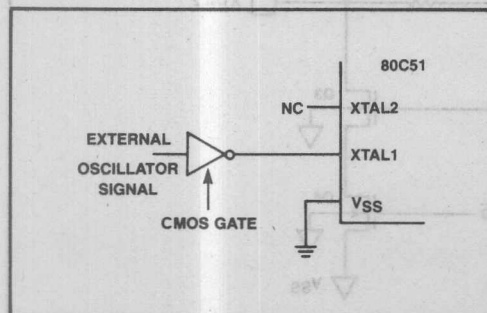


Figure 7-37. Driving the CHMOS MCS-51 Parts with an External Clock Source

## 7.14 INTERNAL TIMING

Figures 7-38 through 7-41 show when the various strobe and port signals are clocked internally. The figures do not show rise and fall times of the signals, nor do they show propagation delays between the XTAL2 signal and events at other pins.

Rise and fall times are dependent on the external loading that each pin must drive. They are often taken to be something in the neighborhood of 10 nsec, measured between 0.8V and 2.0V.

Propagation delays are different for different pins. For a given pin they vary with pin loading, temperature, VCC, and manufacturing lot. If the XTAL2 waveform is taken as the timing reference, prop delays may vary from 25 to 125 nsec.

The AC Timings section of the data sheets do not reference any timing to the XTAL2 waveform. Rather, they relate the critical edges of control and input signals to each other. The timings published in the data sheets include the effects of propagation delays under the specified test conditions.

## 7.15 MCS-51 PIN DESCRIPTIONS

**VCC:** Supply voltage.

**VSS:** Circuit ground potential.

**Port 0:** Port 0 is an 8-bit open drain bidirectional I/O port. As an open drain output port it can sink 8 LS TTL loads. Port 0 pins that have 1s written to them float, and in that state will function as high-impedance inputs. Port 0 is also

the multiplexed low-order address and data bus during accesses to external memory. In this application it uses strong internal pullups when emitting 1s. Port 0 also emits code bytes during program verification. In that application, external pullups are required.

**Port 1:** Port 1 is an 8-bit bidirectional I/O port with internal pullups. The port 1 output buffers can sink/source 4 LS TTL loads. Port 1 pins that have 1s written to them are pulled high by the internal pullups, and in that state can be used as inputs. As inputs, Port 1 pins that are externally being pulled low will source current (IIL, on the data sheet) because of the internal pullups.

In the 8052, pins P1.0 and P1.1 also serve the alternate functions of T2 and T2EX. T2 is the Timer 2 external input. T2EX is the input through which a Timer 2 "capture" is triggered.

**Port 2:** Port 2 is an 8-bit bidirectional I/O port with internal pullups. The Port 2 output buffers can sink/source 4 LS TTL loads. Port 2 emits the high-order address byte during accesses to external memory that use 16-bit addresses. In this application it uses the strong internal pullups when emitting 1s. Port 2 also receives the high-order address and control bits during 8751H programming and verification, and during program verification in the 8051AH.

**Port 3:** Port 3 is an 8-bit bidirectional I/O port with internal pullups. It also serves the functions of various special features of the MCS-51 Family, as listed below:

PORT PIN	ALTERNATE FUNCTION
P3.0	RXD (serial input port)
P3.1	TXD (serial output port)
P3.2	INT0 (external interrupt 0)
P3.3	INT1 (external interrupt 1)
P3.4	T0 (Timer 0 external input)
P3.5	T1 (Timer 1 external input)
P3.6	WR (external data memory write strobe)
P3.7	RD (external data memory read strobe)

The Port 3 output buffers can source/sink 4 LS TTL loads.

**RST:** Reset input. A high on this pin for two machine cycles while the oscillator is running resets the device.

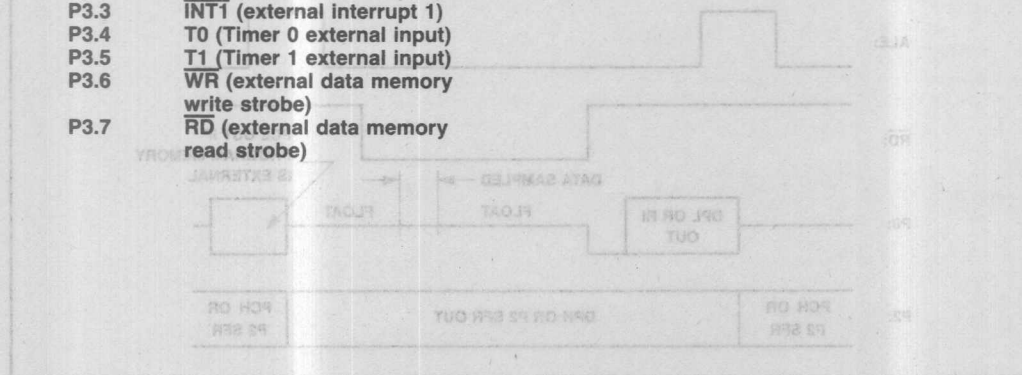
**ALE/PROG:** Address Latch Enable output pulse for latching the low byte of the address during accesses to external memory. ALE is emitted at a constant rate of 1/6 of the oscillator frequency, for external timing or clocking purposes, even when there are no accesses to external memory. (However, one ALE pulse is skipped during each access to external **Data Memory**) This pin is also the program pulse input (**PROG**) during EPROM programming.

**PSEN:** Program Store Enable is the read strobe to external Program Memory. When the device is executing out of external Program Memory, PSEN is activated twice each machine cycle (except that two PSEN activations are skipped during accesses to external **Data Memory**). PSEN is not activated when the device is executing out of internal Program Memory.

**EA/VPP:** When  $\overline{EA}$  is held high the CPU executes out of internal Program Memory (unless the Program Counter exceeds 0FFFH in the 8051AH, or 1FFFH in the 8052). Holding  $\overline{EA}$  low forces the CPU to execute out of external memory regardless of the Program Counter value. In the 8031AH and 8032,  $\overline{EA}$  must be externally wired low. In the 8751H, this pin also receives the 21V programming supply voltage (VPP) during EPROM programming.

**XTAL1:** Input to the inverting oscillator amplifier.

**XTAL2:** Output from the inverting oscillator amplifier.





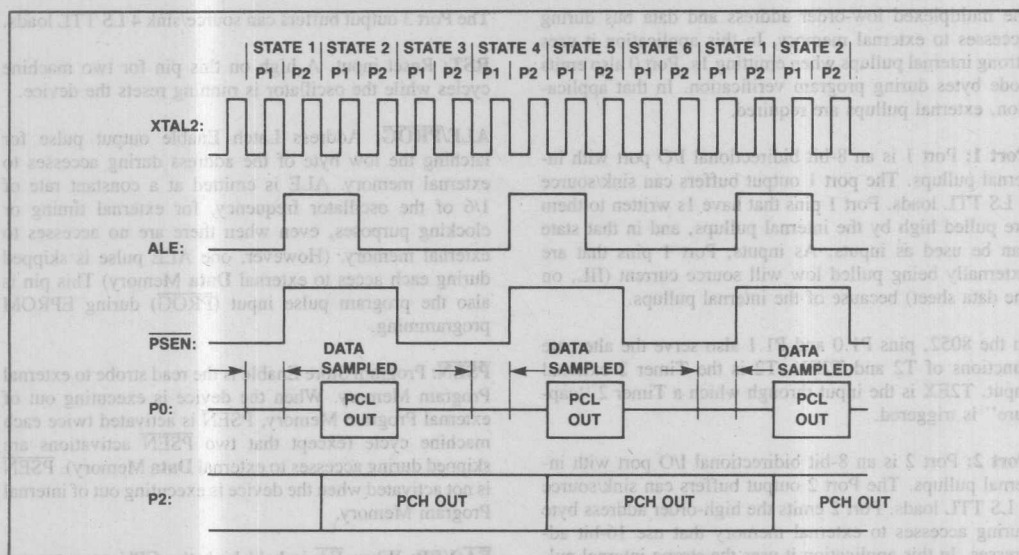


Figure 7-38. External Program Memory Fetches

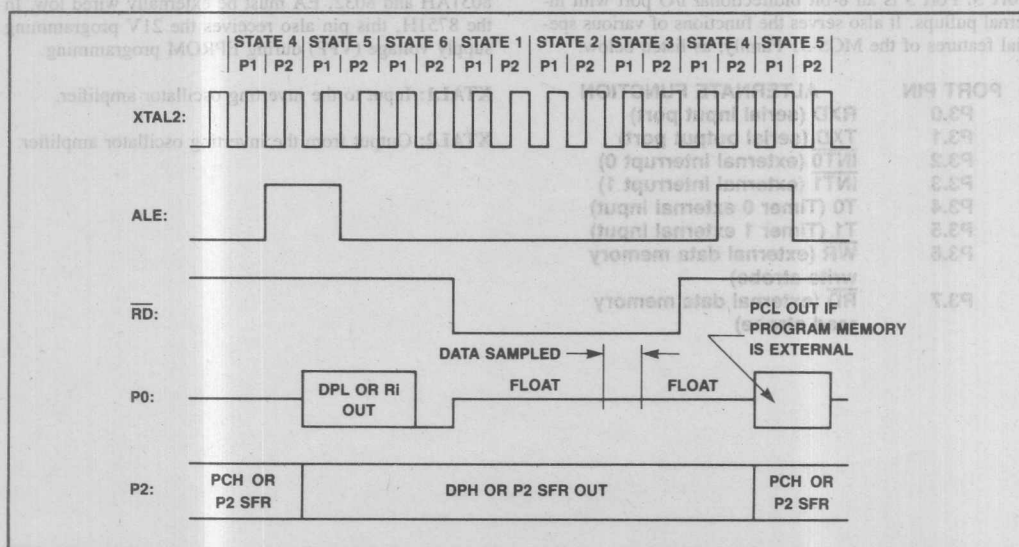


Figure 7-39. External Data Memory Read Cycle

# MCS<sup>®</sup>-51 ARCHITECTURE

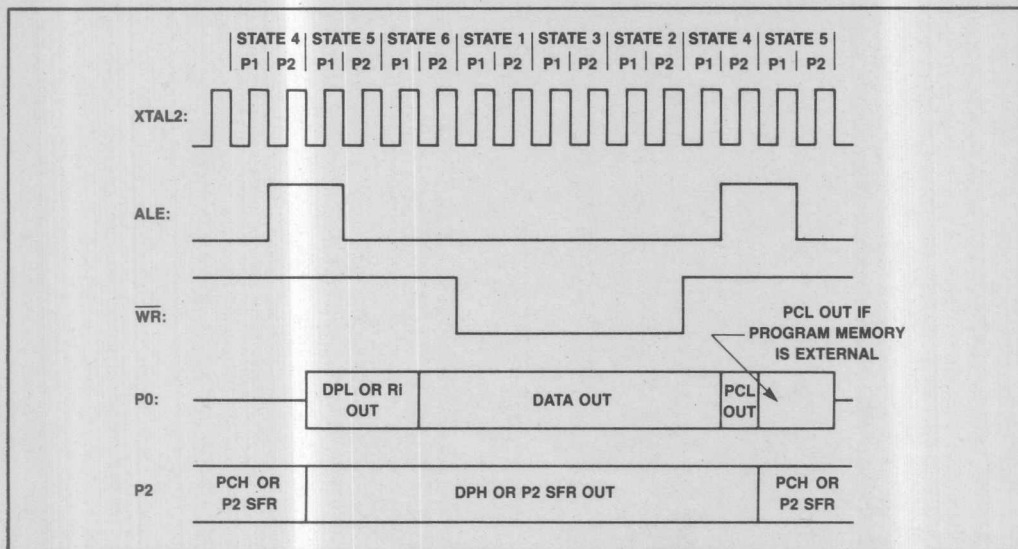


Figure 7-40. External Data Memory Write Cycle

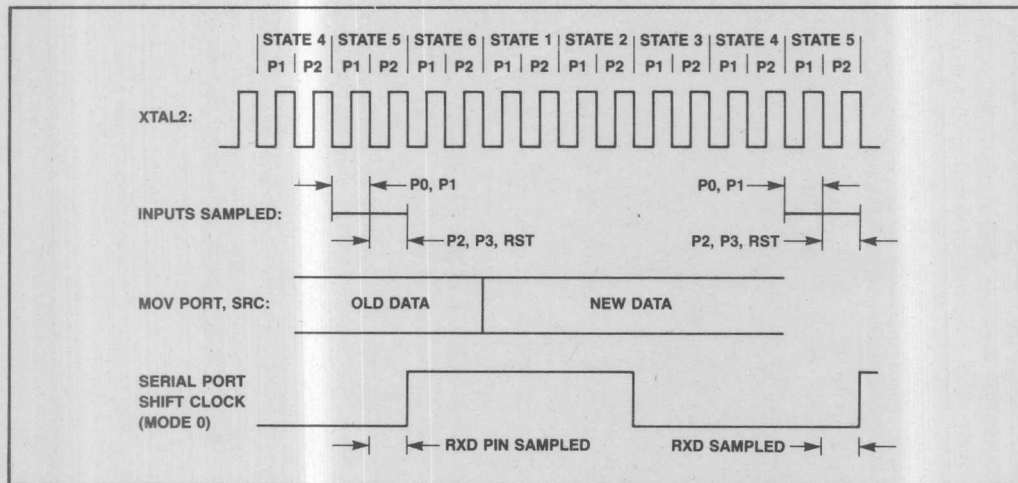


Figure 7-41. Port Operation

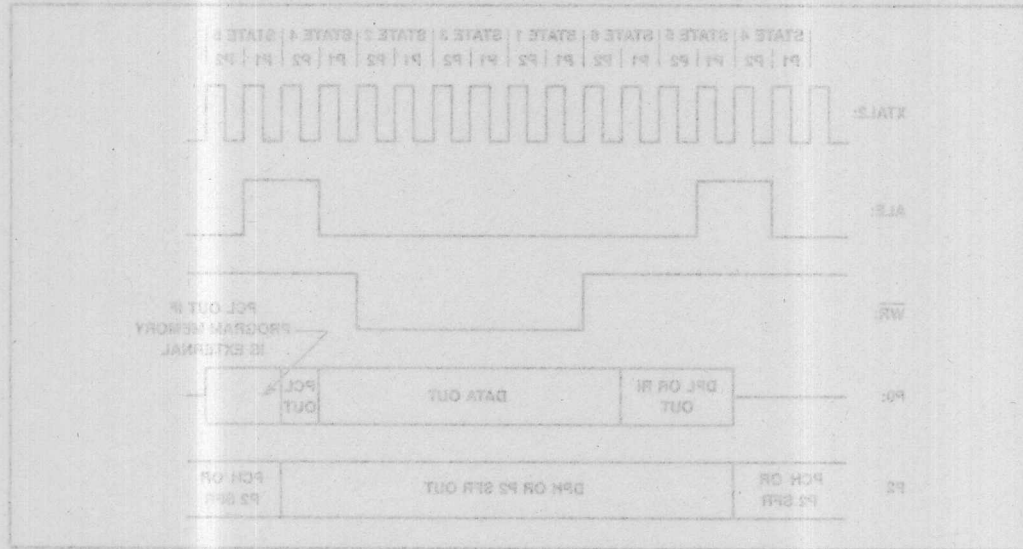


Figure 7-40. External Data Memory Write Cycle

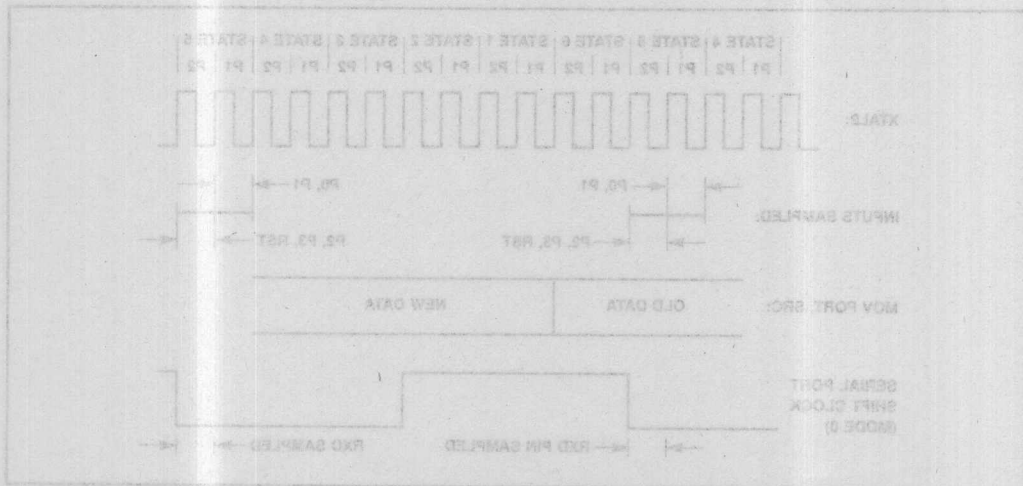


Figure 7-41. Port Operation







# CHAPTER 8

## MCS®-51 INSTRUCTION SET

### 8.0 INTRODUCTION

The MCS®-51 instruction set includes 111 instructions, 49 of which are single-byte, 45 two-byte and 17 three byte. The instruction op code format consists of a function mnemonic followed by a "destination, source" operand field. This field specifies the data type and addressing method(s) to be used.

### 8.1 FUNCTIONAL OVERVIEW

The MCS-51 instruction set is divided into four functional groups:

- Data Transfer
- Arithmetic
- Logic
- Control Transfer

#### 8.1.1 Data Transfer

Data transfer operations are divided into three classes:

- General Purpose
- Accumulator-Specific
- Address-Object

None of these operations affect the PSW flag settings except a POP or MOV directly to the PSW!

#### GENERAL-PURPOSE TRANSFERS

- MOV performs a bit or a byte transfer from the source operand to the destination operand.
- PUSH increments the SP register and then transfers a byte from the source operand to the stack location currently addressed by SP.
- POP transfer a byte operand from the stack location addressed by SP to the destination operand and then decrements SP.

#### ACCUMULATOR SPECIFIC TRANSFERS

- XCH exchanges the byte source operand with register A (accumulator).
- XCHD exchanges the low-order nibble of the byte source operand with the low-order nibble of A.

- MOVX performs a byte move between the External Data Memory and the accumulator. The external address can be specified by the DPTR register (16-bit) or the R1 or R0 register (8-bit).
- MOVC moves a byte from Program memory to the accumulator. The operand in A is used as an index into a 256-byte table pointed to by the base register (DPTR or PC). The byte operand accessed is transferred to the accumulator.

#### ADDRESS-OBJECT TRANSFER

- MOV DPTR, #data loads 16-bits of immediate data into a pair of destination registers, DPH and DPL.

#### 8.1.2 Arithmetic

The 8051 has four basic mathematical operations. Only 8-bit operations using unsigned arithmetic are supported directly. The overflow flag, however, permits the addition and subtraction operation to serve for both unsigned and signed binary integers. Arithmetic can also be performed directly on packed decimal (BCD) representations.

#### ADDITION

- INC (increment) adds one to the source operand and puts the result in the operand.
- ADD adds A to the source operand and returns the result to A.
- ADDC (add with Carry) adds A and the source operand, then adds one (1) if CY is set, and puts the result in A.
- DA (decimal-add-adjust for BCD addition) corrects the sum which results from the binary addition of two two-digit decimal operands. The packed decimal sum formed by DA is returned to A. CY is set if the BCD result is greater than 99; otherwise, it is cleared.

#### SUBTRACTION

- SUBB (subtract with borrow) subtracts the second source operand from the first operand (the accumulator), subtracts one (1) if CY is set and returns the result to A.
- DEC (decrement) subtracts one (1) from the source operand and returns the result to the operand.

## MCS®-51 INSTRUCTION SET

### MULTIPLICATION

- MUL performs an unsigned multiplication of the A register by the B register, returning a double-byte result. A receives the low-order byte, B receives the high-order byte. OV is cleared if the top half of the result is zero and is set if it is non-zero. CY is cleared. AC is unaffected.

### DIVISION

- DIV performs an unsigned division of the A register by the B register and returns the integer quotient to A and returns the fractional remainder to the B register. Division by zero leaves indeterminate data in registers A and B and sets OV; otherwise OV is cleared. CY is cleared. AC is unaffected.

Unless otherwise stated in the above descriptions, the flags of PSW are affected as follows:

- CY is set if the operation causes a carry to or from the resulting high-order bit. Otherwise CY is cleared.
- AC is set if the operation results in a carry from the low-order four bits of the result (during addition), or a borrow from the high-order bits to the low-order bits (during subtraction); otherwise AC is cleared.
- OV is set if the operation results in a carry to the high-order bit of the result but not a carry from the high-order bit, or vice versa; otherwise OV is cleared. OV is used in two's-complement arithmetic, because it is set when the signed result cannot be represented in 8 bits.
- P is set if the modulo 2 sum of the eight bits in the accumulator is 1 (odd parity); otherwise P is cleared (even parity). When a value is written to the PSW register, the P bit remains unchanged, as it always reflects the parity of A.

### 8.1.3 Logic

The 8051 performs basic logic operations on both bit and byte operands.

### SINGLE-OPERAND OPERATIONS

- CLR sets A or any directly addressable bit to zero (0).
- SETB sets any directly addressable bit to one (1).
- CPL is used to complement the contents of the A register without affecting any flags, or any directly addressable bit location.

- RL, RLC, RR, RRC, SWAP are the five rotate operations that can be performed on A. RL, rotate left, RR, rotate right, RLC, rotate left through C, RRC, rotate right through C, and SWAP, rotate left four. For RLC and RRC the CY flag becomes equal to the last bit rotated out. SWAP rotates A left four places to exchange bits 3 through 0 with bits 7 through 4.

### TWO-OPERAND OPERATIONS

- ANL performs bitwise logical and of with two source operands (for both bit and byte operands) and returns the result to the location of the first operand.
- ORL performs bitwise logical or of two source operands (for both bit and byte operands) and returns the result of the location of the first operand.
- XRL performs bitwise logical xor of two source operands (byte operands) and returns the result to the location of the first operand.

### 8.1.4 Control Transfer

There are three classes of control transfer operations: unconditional calls, returns and jumps; conditional jumps; and interrupts. All control transfer operations cause, some upon a specific condition, the program execution to continue at a non-sequential location in program memory.

### UNCONDITIONAL CALLS, RETURNS AND JUMPS

Unconditional calls, returns and jumps transfer control from the current value of the Program Counter to the target address. Both direct and indirect transfers are supported.

- ACALL and LCALL push the address of the next instruction onto the stack and then transfer control to the target address. ACALL is a 2-byte instruction used when the target address is in the current 2K page. LCALL is a 3-byte instruction that addresses the full 64K program space. In ACALL, immediate data (i.e. an 11 bit address field) is concatenated to the five most significant bits of the PC (which is pointing to the next instruction). If ACALL is in the last 2 bytes of a 2K page then the call will be made to the next page since the PC will have been incremented to the next instruction prior to execution.

- RET transfers control to the return address saved on the stack by a previous call operation and decrements the SP register by two (2) to adjust the SP for the popped address.
- AJMP, LJMP and SJMP transfer control to the target operand. The operation of AJMP and LJMP are analogous to ACALL and LCALL. The SJMP (short jump) instruction provides for transfers within a 256 byte range centered about the starting address of the next instruction (-128 to +127).
- JMP @A+DPTR performs a jump relative to the DPTR register. The operand in A is used as the offset (0-255) to the address in the DPTR register. Thus, the effective destination for a jump can be anywhere in the Program Memory space.

### CONDITIONAL JUMPS

Conditional jumps perform a jump contingent upon a specific condition. The destination will be within a 256-byte range centered about the starting address of the next instruction (-128 to +127).

- JZ performs a jump if the accumulator is zero.
- JNZ performs a jump if the accumulator is not zero.
- JC performs a jump if the carry flag is set.
- JNC performs a jump if the carry flag is not set.
- JB performs a jump if the Direct Addressed bit is set.
- JNB performs a jump if the Direct Addressed bit is not set.
- JBC performs a jump if the Direct Addressed bit is set and then clears the Direct Addressed bit.
- CJNE compares the first operand to the second operand and performs a jump if they are not equal. CY is set if the first operand is less than the second operand; otherwise it is cleared. Comparisons can be

made between A directly addressable bytes in Internal Data Memory or between an immediate value and either A, a register in the selected Register Bank, or a Register-Indirect addressed byte of Internal RAM.

- DJNZ decrements the source operand and returns the result to the operand. A jump is performed if the result is not zero. The source operand of the DJNZ instruction may be any byte in the Internal Data Memory. Either Direct or Register Addressing may be used to address the source operand.

### INTERRUPT RETURNS

- RETI transfers control as does RET, but additionally enables interrupts of the current priority level.

### 8.2 INSTRUCTION DEFINITIONS

Each of the 51 basic MCS-51 operations, ordered alphabetically according to the operation mnemonic are described beginning page 8-8.

A brief example of how the instruction might be used is given as well as its effect on the PSW flags. The number of bytes and machine cycles required, the binary machine-language encoding, and a symbolic description or restatement of the function is also provided.

Note: Only the carry, auxiliary-carry, and overflow flags are discussed. The parity bit is computed after every instruction cycle that alters the accumulator. Similarly, instructions which alter directly addressed registers could affect the other status flags if the instruction is applied to the PSW. Status flags can also be modified by bit-manipulation.

For details on the MCS-51 assembler, ASM51, refer to the MCS-51 Macro Assembler User's Guide, publication number 9800937.

Table 8-1 summarized the MCS-51 instruction set.



Table 8-1. 8051 Instruction Set Summary

Interrupt Response Time: To finish execution of current instruction, respond to the interrupt request, push the PC and to vector to the first instruction of the interrupt service program requires 38 to 81 oscillator periods (3 to 7  $\mu$ s @ 12 MHz).

### INSTRUCTIONS THAT AFFECT FLAG SETTINGS<sup>1</sup>

INSTRUCTION	FLAG	INSTRUCTION	FLAG
	C OV AC		C OV AC
ADD	X X X	CLR C	O
ADDC	X X X	CPL C	X
SUBB	X X X	ANL C, bit	X
MUL	O X	ANL C, /bit	X
DIV	O X	ORL C, bit	X
DA	X	ORL C, bit	X
RRC	X	MOV C, bit	X
RLC	X	CJNE	X
SETB C	1		

<sup>1</sup>Note that operations on SFR, byte address 208 or bit addresses 209-215 (i.e., the PSW or bits in the PSW) will also affect flag settings.

Notes on instruction set and addressing modes:

- Rn —Register R7-R0 of the currently selected Register Bank.
- direct —8-bit internal data location's address. This could be an Internal Data RAM location (0-127) or a SFR [i.e., I/O port, control register, status register, etc. (128-255)].
- @Ri —8-bit internal data RAM location (0-255) addressed indirectly through register R1 or R0.
- #data —8-bit constant included in instruction.
- #data 16 —16-bit constant included in instruction.
- addr 16 —16-bit destination address. Used by LCALL & LJMP. A branch can be anywhere within the 64K-byte Program Memory address space.
- addr 11 —11-bit destination address. Used by ACALL & AJMP. The branch will be within the same 2K-byte page of program memory as the first byte of the following instruction.
- rel —Signed (two's complement) 8-bit offset byte. Used by SJMP and all conditional jumps. Range is -128 to +127 bytes relative to first byte of the following instruction.
- bit —Direct Addressed bit in Internal Data RAM or Special Function Register.
- \* —New operation not provided by 8048AH/8049AH.

### ARITHMETIC OPERATIONS

Mnemonic	Description	Byte	Oscillator Period
ADD A, Rn	Add register to Accumulator	1	12
ADD A, direct	Add direct byte to Accumulator	2	12
ADD A, @Ri	Add indirect RAM to Accumulator	1	12
ADD A, #data	Add immediate data to Accumulator	2	12
ADDC A, Rn	Add register to Accumulator with Carry	1	12
ADDC A, direct	Add direct byte to Accumulator with Carry	2	12
ADDC A, @Ri	Add indirect RAM to Accumulator with Carry	1	12
ADDC A, #data	Add immediate data to Acc with Carry	2	12
SUBB A, Rn	Subtract register from Acc with borrow	1	12
SUBB A, direct	Subtract direct byte from Acc with borrow	2	12

### ARITHMETIC OPERATIONS Cont.

Mnemonic	Description	Byte	Oscillator Period
SUBB A, @Ri	Subtract indirect RAM from Acc with borrow	1	12
SUBB A, #data	Subtract immediate data from Acc with borrow	2	12
INC A	Increment Accumulator	1	12
INC Rn	Increment register	1	12
INC direct	Increment direct byte	2	12
INC @Ri	Increment indirect RAM	1	12
DEC A	Decrement Accumulator	1	12
DEC Rn	Decrement Register	1	12
DEC direct	Decrement direct byte	2	12
DEC @Ri	Decrement indirect RAM	1	12
INC DPTR	Increment Data Pointer	1	24
MUL AB	Multiply A & B	1	48
DIV AB	Divide A by B	1	48
DA A	Decimal Adjust Accumulator	1	12

All mnemonics copyrighted ©Intel Corporation 1980

Table 8-1. 8051 Instruction Set Summary (Continued)

LOGICAL OPERATIONS				
Mnemonic	Description	Byte	Oscillator Period	
ANL A,Rn	AND register to Accumulator	1	12	
ANL A,direct	AND direct byte to Accumulator	2	12	
ANL A,@Ri	AND indirect RAM to Accumulator	1	12	
ANL A,#data	AND immediate data to Accumulator	2	12	
ANL direct,A	AND Accumulator to direct byte	2	12	
ANL direct,#data	AND immediate data to direct byte	3	24	
ORL A,Rn	OR register to Accumulator	1	12	
ORL A,direct	OR direct byte to Accumulator	2	12	
ORL A,@Ri	OR indirect RAM to Accumulator	1	12	
ORL A,#data	OR immediate data to Accumulator	2	12	
ORL direct,A	OR Accumulator to direct byte	2	12	
ORL direct,#data	OR immediate data to direct byte	3	24	
XRL A,Rn	Exclusive-OR register to Accumulator	1	12	
XRL A,direct	Exclusive-OR direct byte to Accumulator	2	12	

LOGICAL OPERATIONS Cont.				
Mnemonic	Description	Byte	Oscillator Period	
XRL A,@Ri	Exclusive-OR indirect RAM to Accumulator	1	12	
XRL A,#data	Exclusive-OR immediate data to Accumulator	2	12	
XRL direct,A	Exclusive-OR Accumulator to direct byte	2	12	
XRL direct,#data	Exclusive-OR immediate data to direct byte	3	24	
CLR A	Clear Accumulator	1	12	
CPL A	Complement Accumulator	1	12	
RL A	Rotate Accumulator Left	1	12	
RLC A	Rotate Accumulator Left through the Carry	1	12	
RR A	Rotate Accumulator Right	1	12	
RRC A	Rotate Accumulator Right through the Carry	1	12	
SWAP A	Swap nibbles within the Accumulator	1	12	

All mnemonics copyrighted © Intel Corporation 1980

# MCS®-51 INSTRUCTION SET

Table 8-1. 8051 Instruction Set Summary (Continued)

DATA TRANSFER					DATA TRANSFER Cont.				
Mnemonic	Description	Byte	Oscillator Period		Mnemonic	Description	Byte	Oscillator Period	
MOV A,Rn	Move register to Accumulator	1	12		MOV DPTR,#data16	Load Data Pointer with a 16-bit constant	3	24	
MOV A,direct	Move direct byte to Accumulator	2	12		MOVC A,@A+DPTR	Move Code byte relative to DPTR to Acc	1	24	
MOV A,@Ri	Move indirect RAM to Accumulator	1	12		MOVC A,@A+PC	Move Code byte relative to PC to Acc	1	24	
MOV A,#data	Move immediate data to Accumulator	2	12		MOVX A,@Ri	Move External RAM (8-bit addr) to Acc	1	24	
MOV Rn,A	Move Accumulator to register	1	12		MOVX A,@DPTR	Move External RAM (16-bit addr) to Acc	1	24	
MOV Rn,direct	Move direct byte to register	2	24		MOVX @Ri,A	Move Acc to External RAM (8-bit addr)	1	24	
MOV Rn,#data	Move immediate data to register	2	12		MOVX @DPTR,A	Move Acc to External RAM (16-bit addr)	1	24	
MOV direct,A	Move Accumulator to direct byte	2	12		PUSH direct	Push direct byte onto stack	2	24	
MOV direct,Rn	Move register to direct byte	2	24		POP direct	Pop direct byte from stack	2	24	
MOV direct,direct	Move direct byte to direct	3	24		XCH A,Rn	Exchange register with Accumulator	1	12	
MOV direct,@Ri	Move indirect RAM to direct byte	2	24		XCH A,direct	Exchange direct byte with Accumulator	2	12	
MOV direct,#data	Move immediate data to direct byte	3	24		XCH A,@Ri	Exchange indirect RAM with Accumulator	1	12	
MOV @Ri,A	Move Accumulator to indirect RAM	1	12		XCHD A,@Ri	Exchange low-order Digit indirect RAM with Acc	1	12	
MOV @Ri,direct	Move direct byte to indirect RAM	2	24						
MOV @Ri,#data	Move immediate data to indirect RAM	2	12						

All mnemonics copyrighted ©Intel Corporation 1980

# MCS®-51 INSTRUCTION SET

Table 8-1. 8051 Instruction Set Summary (Continued)

BOOLEAN VARIABLE MANIPULATION					PROGRAM BRANCHING Cont.				
Mnemonic	Description	Byte	Oscillator Period		Mnemonic	Description	Byte	Oscillator Period	
CLR C	Clear Carry	1	12		RET	Return from interrupt	1	24	
CLR bit	Clear direct bit	2	12		AJMP addr11	Absolute Jump	2	24	
SETB C	Set Carry	1	12		LJMP addr16	Long Jump	3	24	
SETB bit	Set direct bit	2	12		SJMP rel	Short Jump (relative addr)	2	24	
CPL C	Complement Carry	1	12		JMP @A+DPTR	Jump indirect relative to the DPTR	1	24	
CPL bit	Complement direct bit	2	12		JZ rel	Jump if Accumulator is Zero	2	24	
ANL C,bit	AND direct bit to Carry	2	24		JNZ rel	Jump if Accumulator is Not Zero	2	24	
ANL C,/bit	AND complement of direct bit to Carry	2	24		CJNE A,direct,rel	Compare direct byte to Acc and Jump if Not Equal	3	24	
ORL C,bit	OR direct bit to Carry	2	24		CJNE A,#data,rel	Compare immediate to Acc and Jump if Not Equal	3	24	
ORL C,/bit	OR complement of direct bit to Carry	2	24		CJNE Rn,#data,rel	Compare immediate to register and Jump If Not Equal	3	24	
MOV C,bit	Move direct bit to Carry	2	12		CJNE @Ri,#data,rel	Compare immediate to indirect and Jump if Not Equal	3	24	
MOV bit,C	Move Carry to direct bit	2	24		DJNZ Rn,rel	Decrement register and Jump if Not Zero	3	24	
JC rel	Jump if Carry is set	2	24		DJNZ direct,rel	Decrement direct byte and Jump if Not Zero	3	24	
JNC rel	Jump if Carry not set	2	24		NOP	No Operation	1	12	
JB bit,rel	Jump if direct Bit is set	3	24						
JNB bit,rel	Jump if direct Bit is Not set	3	24						
JBC bit,rel	Jump if direct Bit is set & clear bit	3	24						

PROGRAM BRANCHING				
Mnemonic	Description	Byte	Oscillator Period	
ACALL addr11	Absolute Subroutine Call	2	24	
LCALL addr16	Long Subroutine Call	3	24	
RET	Return from Subroutine	1	24	

All mnemonics copyrighted ©Intel Corporation 1980



ACALL      addr11

**Function:** Absolute Call

**Description:** ACALL unconditionally calls a subroutine located at the indicated address. The instruction increments the PC twice to obtain the address of the following instruction, then pushes the 16-bit result onto the stack (low-order byte first) and increments the stack pointer twice. The destination address is obtained by successively concatenating the five high-order bits of the incremented PC, op-code bits 7-5, and the second byte of the instruction. The subroutine called must therefore start within the same 2K block of the program memory as the first byte of the instruction following ACALL. No flags are affected.

**Example:** Initially SP equals 07H. The label "SUBRTN" is at program memory location 0345H. After executing the instruction,

ACALL      SUBRTN

at location 0123H, SP will contain 09H, internal RAM locations 08H and 09H will contain 25H and 01H, respectively, and the PC will contain 0345H.

**Bytes:** 2  
**Cycles:** 2

**Encoding:**      a10   a9   a8   1   0   0   0   1

                         a7   a6   a5   a4   a3   a2   a1   a0

**Operation:** ACALL  
(PC) ← (PC) + 2  
(SP) ← (SP) + 1  
((SP)) ← (PC7-0)  
(SP) ← (SP) + 1  
((SP)) ← (PC15-8)  
(PC10-0) ← page address

PROGRAM BRANCHING			
Mnemonic	Description	Byte	Period
ACALL addr11	Absolute Subroutine Call	2	24
LCALL addr16	Long Subroutine Call	3	24
RET	Return from Subroutine	1	24

## ADD A,<src-byte>

**Function:** Add

**Description:** ADD adds the byte variable indicated to the accumulator, leaving the result in the accumulator. The carry and auxiliary-carry flags are set, respectively, if there is a carry-out from bit 7 or bit 3, and cleared otherwise. When adding unsigned integers, the carry flag indicates an overflow occurred.

OV is set if there is a carry-out of bit 6 but not out of bit 7, or a carry-out of bit 7 but not bit 6; otherwise OV is cleared. When adding signed integers, OV indicates a negative number produced as the sum of two positive operands, or a positive sum from two negative operands.

Four source operand addressing modes are allowed: register, direct, register-indirect, or immediate.

**Example:** The accumulator holds 0C3H (11000011B) and register 0 holds 0AAH (10101010B). The instruction,

ADD A,R0

will leave 6DH (01101101B) in the accumulator with the AC flag cleared and both the carry flag and OV set to 1.

### ADD A,Rn

**Bytes:** 1

**Cycles:** 1

**Encoding:**

0	0	1	0	1	r	r	r
---	---	---	---	---	---	---	---

**Operation:** ADD  
 $(A) \leftarrow (A) + (Rn)$

### ADD A,direct

**Bytes:** 2

**Cycles:** 1

**Encoding:**

0	0	1	0	0	1	0	1
---	---	---	---	---	---	---	---

direct address
----------------

**Operation:** ADD  
 $(A) \leftarrow (A) + (\text{direct})$

ADD A,@Ri												
Bytes:	1											
Cycles:	1											
Encoding:	<table><tr><td>0</td><td>0</td><td>1</td><td>0</td><td>0</td><td>1</td><td>1</td><td>i</td></tr></table>	0	0	1	0	0	1	1	i			
0	0	1	0	0	1	1	i					
Operation:	ADD (A)←(A) + ((Ri))											
ADD A,#data												
Bytes:	2											
Cycles:	1											
Encoding:	<table><tr><td>0</td><td>0</td><td>1</td><td>0</td><td>0</td><td>1</td><td>0</td><td>0</td></tr></table>	0	0	1	0	0	1	0	0	<table><tr><td>immediate data</td></tr></table>	immediate data	
0	0	1	0	0	1	0	0					
immediate data												
Operation:	ADD (A)←(A) + #data											

ADD A,R0	ADD A,direct	Function: Add	Description: ADD adds the byte variable indicated in the accumulator, saving the result in the accumulator. The carry and flags are not affected. When adding unsigned integers, the carry flag indicates an overflow or a negative sum from two negative operands.																
Bytes: 1	Bytes: 2																		
Cycles: 1	Cycles: 1																		
Encoding:	Encoding:																		
<table><tr><td>0</td><td>0</td><td>1</td><td>0</td><td>1</td><td>1</td><td>1</td><td>r</td></tr></table>	0	0	1	0	1	1	1	r	<table><tr><td>0</td><td>0</td><td>1</td><td>0</td><td>0</td><td>1</td><td>0</td><td>1</td></tr></table>	0	0	1	0	0	1	0	1		
0	0	1	0	1	1	1	r												
0	0	1	0	0	1	0	1												
Operation:	Operation:																		
ADD (A) ← (A) + (Rn)	ADD (A) ← (A) + (direct)																		

**ADDC A, <src-byte>****Function:** Add with Carry

**Description:** ADDC simultaneously adds the byte variable indicated; the carry flag and the accumulator contents, leaving the result in the accumulator. The carry and auxiliary-carry flags are set, respectively, if there is a carry-out from bit 7 or bit 3, and cleared otherwise. When adding unsigned integers, the carry flag indicates an overflow occurred.

OV is set if there is a carry-out of bit 6 but not out of bit 7, or a carryout of bit 7 but not out of bit 6; otherwise OV is cleared. When adding signed integers, OV indicates a negative number produced as the sum of two positive operands or a positive sum from two negative operands.

Four source operand addressing modes are allowed: register, direct, register-indirect, or immediate.

**Example:** The accumulator holds 0C3H (11000011B) and register 0 holds 0AAH (10101010B) with the carry flag set. The instruction,

```
ADDC A,R0
```

will leave 6EH (01101110B) in the accumulator with AC cleared and both the carry flag and OV set to 1.

**ADDC A,Rn****Bytes:** 1**Cycles:** 1**Encoding:**

0	0	1	1	1	r	r	r	r
---	---	---	---	---	---	---	---	---

**Operation:**

ADDC

$$(A) \leftarrow (A) + (C) + (R_n)$$
**ADDC A,direct****Bytes:** 2**Cycles:** 1**Encoding:**

0	0	1	1	0	1	0	1
---	---	---	---	---	---	---	---

direct address

**Operation:**

ADDC

$$(A) \leftarrow (A) + (C) + (\text{direct})$$



**ADDC A,@Ri****Bytes:** 1**Cycles:** 1**Encoding:** 0 0 1 1 0 1 1 i**Operation:** ADDC  
(A) ← (A) + (C) + ((Ri))**ADDC A,#data****Bytes:** 2**Cycles:** 1**Encoding:** 0 0 1 1 0 1 0 0 immediate data**Operation:** ADDC  
(A) ← (A) + (C) + #data**AJMP addr11****Function:** Absolute Jump**Description:** AJMP transfers program execution to the indicated address, which is formed at run-time by concatenating the high-order five bits of the PC (*after* incrementing the PC twice), opcode bits 7-5, and the second byte of the instruction. The destination must therefore be within the same 2K block of program memory as the first byte of the instruction following AJMP.**Example:** The label "JMPADR" is at program memory location 0123H. The instruction, AJMP JMPADR

is at location 0345H and will load the PC with 0123H.

**Bytes:** 2**Cycles:** 2**Encoding:** a10 a9 a8 0 0 0 0 1 a7 a6 a5 a4 a3 a2 a1 a0**Operation:** AJMP  
(PC) ← (PC) + 2  
(PC10-0) ← page address

**ANL**    <dest-byte> , <src-byte>**Function:** Logical-AND for byte variables**Description:** ANL performs the bitwise logical-AND operation between the variables indicated and stores the results in the destination variable. No flags are affected.

The two operands allow six addressing mode combinations. When the destination is the accumulator, the source can use register, direct, register-indirect, or immediate addressing; when the destination is a direct address, the source can be the accumulator or immediate data.

*Note:* When this instruction is used to modify an output port, the value used as the original port data will be read from the output data latch, *not* the input pins.

**Example:** If the accumulator holds 0C3H (11000011B) and register 0 holds 0AAH (10101010B) then the instruction,

```
ANL    A,R0
```

will leave 41H (01000001B) in the accumulator.

When the destination is a directly addressed byte, this instruction will clear combinations of bits in any RAM location or hardware register. The mask byte determining the pattern of bits to be cleared would either be a constant contained in the instruction or a value computed in the accumulator at run-time. The instruction,

```
ANL    P1,#01110011B
```

will clear bits 7, 3, and 2 of output port 1.

**ANL A,Rn****Bytes:** 1**Cycles:** 1**Encoding:**

0	1	0	1	1	r	r	r
---	---	---	---	---	---	---	---

**Operation:**

```
ANL
(A) ← (A) ^ (Rn)
```

**ANL A,direct****Bytes:** 2**Cycles:** 1**Encoding:**

0	1	0	1	0	1	0	1
---	---	---	---	---	---	---	---

direct address

**Operation:**

ANL

 $(A) \leftarrow (A) \wedge (\text{direct})$ **ANL A,@Ri****Bytes:** 1**Cycles:** 1**Encoding:**

0	1	0	1	0	1	1	i
---	---	---	---	---	---	---	---

**Operation:**

ANL

 $(A) \leftarrow (A) \wedge ((Ri))$ **ANL A,#data****Bytes:** 2**Cycles:** 1**Encoding:**

0	1	0	1	0	1	0	0
---	---	---	---	---	---	---	---

immediate data

**Operation:**

ANL

 $(A) \leftarrow (A) \wedge \#data$ **ANL direct,A****Bytes:** 2**Cycles:** 1**Encoding:**

0	1	0	1	0	0	1	0
---	---	---	---	---	---	---	---

direct address

**Operation:**

ANL

 $(\text{direct}) \leftarrow (\text{direct}) \wedge (A)$ **ANL direct,#data****Bytes:** 3**Cycles:** 2**Encoding:**

0	1	0	1	0	0	1	1
---	---	---	---	---	---	---	---

direct address

immediate data

**Operation:**

ANL

 $(\text{direct}) \leftarrow (\text{direct}) \wedge \#data$

**ANL C, <src-bit>****Function:** Logical-AND for bit variables**Description:** If the Boolean value of the source bit is a logical 0 then clear the carry flag; otherwise leave the carry flag in its current state. A slash ("/") preceding the operand in the assembly language indicates that the logical complement of the addressed bit is used as the source value, *but the source bit itself is not affected*. No other flags are affected.

Only direct bit addressing is allowed for the source operand.

**Example:** Set the carry flag if, and only if, P1.0=1, ACC. 7=1, and OV=0:

```

MOV C,P1.0      ;LOAD CARRY WITH INPUT PIN STATE
ANL C,ACC.7     ;AND CARRY WITH ACCUM. BIT 7
ANL C,/OV       ;AND WITH INVERSE OF OVERFLOW
                  FLAG

```

**ANL C,bit****Bytes:** 2**Cycles:** 2**Encoding:**

1	0	0	0	0	0	1	0
---	---	---	---	---	---	---	---

bit address

**Operation:**

$$\text{ANL} \\ (C) \leftarrow (C) \wedge (\text{bit})$$
**ANL C,/bit****Bytes:** 2**Cycles:** 2**Encoding:**

1	0	1	1	0	0	0	0
---	---	---	---	---	---	---	---

bit address

**Operation:**

$$\text{ANL} \\ (C) \leftarrow (C) \wedge \neg (\text{bit})$$
**CJNE <dest-byte>, <src-byte>, rel****Function:** Compare and Jump if Not Equal.**Description:** CJNE compares the magnitudes of the first two operands, and branches if their values are not equal. The branch destination is computed by adding the signed relative-displacement in the last instruction byte to the PC, after incrementing the PC to the start of the next instruction. The carry flag is set if the unsigned integer value of <dest-byte> is less than the unsigned integer value of <src-byte>; otherwise, the carry is cleared. Neither operand is affected.



The first two operands allow four addressing mode combinations: the accumulator may be compared with any directly addressed byte or immediate data, and any indirect RAM location or working register can be compared with an immediate constant.

**Example:** The accumulator contains 34H. Register 7 contains 56H. The first instruction in the sequence,

```
CJNE    R7,#60H, NOT_EQ
;      ...      ; R7 = 60H.
NOT_EQ: JC    REQ_LOW      ; IF R7 < 60H.
;      ...      ; R7 > 60H.
```

sets the carry flag and branches to the instruction at label NOT\_EQ. By testing the carry flag, this instruction determines whether R7 is greater or less than 60H.

If the data being presented to port 1 is also 34H, then the instruction,

```
WAIT: CJNE A,P1,WAIT
```

clears the carry flag and continues with the next instruction in sequence, since the accumulator does equal the data read from P1. (If some other value was being input on P1, the program will loop at this point until the P1 data changes to 34H.)

**CJNE A,direct,rel**

**Bytes:** 3

**Cycles:** 2

**Encoding:**

1	0	1	1	0	1	0	1	direct address	rel. address
---	---	---	---	---	---	---	---	----------------	--------------

**Operation:**

(PC) ← (PC) + 3

IF (A) <> (direct)

THEN

(PC) ← (PC) + relative offset

IF (A) < (direct)

THEN

(C) ← 1

ELSE

(C) ← 0

# MCS®-51 INSTRUCTION SET

## CJNE A,#data,rel

Bytes: 3

Cycles: 2

Encoding: 

1	0	1	1	0	1	0	0
---	---	---	---	---	---	---	---

immediate data rel. address

**Operation:** (PC)  $\leftarrow$  (PC) + 3  
 IF (A)  $\neq$  data  
 THEN  
     (PC)  $\leftarrow$  (PC) + relative offset  
 IF (A) < data  
 THEN  
     (C)  $\leftarrow$  1  
 ELSE  
     (C)  $\leftarrow$  0

## CJNE Rn,#data,rel

Bytes: 3

Cycles: 2

Encoding: 

1	0	1	1	1	r	r	r
---	---	---	---	---	---	---	---

immediate data rel. address

**Operation:** (PC)  $\leftarrow$  (PC) + 3  
 IF (Rn)  $\neq$  data  
 THEN  
     (PC)  $\leftarrow$  (PC) + relative offset  
 IF (Rn) < data  
 THEN  
     (C)  $\leftarrow$  1  
 ELSE  
     (C)  $\leftarrow$  0

## CJNE @Ri,#data,rel

Bytes: 3

Cycles: 2

Encoding: 

1	0	1	1	0	1	1	i
---	---	---	---	---	---	---	---

immediate data rel. address

**Operation:** (PC)  $\leftarrow$  (PC) + 3  
 IF ((Ri))  $\neq$  data  
 THEN  
     (PC)  $\leftarrow$  (PC) + relative offset  
 IF ((Ri)) < data  
 THEN  
     (C)  $\leftarrow$  1  
 ELSE  
     (C)  $\leftarrow$  0

**CLR A****Function:** Clear Accumulator**Description:** The accumulator is cleared (all bits set to zero). No flags are affected.**Example:** The accumulator contains 5CH (01011100B). The instruction,

CLR A

will leave the accumulator set to 00H (00000000B).

**Bytes:** 1**Cycles:** 1**Encoding:**

1	1	1	0	0	1	0	0
---	---	---	---	---	---	---	---

**Operation:** CLR  
(A) ← 0**CLR bit****Function:** Clear bit**Description:** The indicated bit is cleared (reset to zero). No other flags are affected. CLR can operate on the carry flag or any directly addressable bit.**Example:** Port 1 has previously been written with 5DH (01011101B). The instruction,

CLR P1.2

will leave the port set to 59H (01011001B).

**CLR C****Bytes:** 1**Cycles:** 1**Encoding:**

1	1	0	0	0	0	1	1
---	---	---	---	---	---	---	---

**Operation:** CLR  
(C) ← 0**CLR bit****Bytes:** 2**Cycles:** 1**Encoding:**

1	1	0	0	0	0	1	0
---	---	---	---	---	---	---	---

bit address
-------------

**Operation:** CLR  
(bit) ← 0

**CPL    A****Function:** Complement Accumulator**Description:** Each bit of the accumulator is logically complemented (one's complement). Bits which previously contained a one are changed to zero and vice-versa. No flags are affected.**Example:** The accumulator contains 5CH (01011100B). The instruction,

CPL    A

will leave the accumulator set to 0A3H (10100011B).

**Bytes:** 1**Cycles:** 1**Encoding:**

1	1	1	1	0	1	0	0
---	---	---	---	---	---	---	---

**Operation:** CPL  
(A) ← ¬(A)**CPL    bit****Function:** Complement bit**Description:** The bit variable specified is complemented. A bit which had been a one is changed to zero and vice-versa. No other flags are affected. CLR can operate on the carry or any directly addressable bit.**Example:** *Note:* When this instruction is used to modify an output pin, the value used as the original data will be read from the output data latch, *not* the input pin. Port 1 has previously been written with 5BH (01011101B). The instruction sequence,

CPL    P1.1

CPL    P1.2

will leave the port set to 5BH (01011101B).

**CPL    C****Bytes:** 1**Cycles:** 1**Encoding:**

1	0	1	1	0	0	1	1
---	---	---	---	---	---	---	---

**Operation:** CPL

(C) ← ¬(C)



# MCS®-51 INSTRUCTION SET

**CPL bit**

**Bytes:** 2

**Cycles:** 1

**Encoding:**

1	0	1	1	0	0	1	0
---	---	---	---	---	---	---	---

bit address

### Operation:

CPL

$$(\text{bit}) \leftarrow \neg (\text{bit})$$

DA A

**Function:**

### Decimal-adjust Accumulator for Addition

**Description:**

DA A adjusts the eight-bit value in the accumulator resulting from the earlier addition of two variables (each in packed-BCD format), producing two four-bit digits. Any ADD or ADDC instruction may have been used to perform the addition.

If accumulator bits 3-0 are greater than nine (xxxx1010-xxxx1111), or if the AC flag is one, six is added to the accumulator producing the proper BCD digit in the low-order nibble. This internal addition would set the carry flag if a carry-out of the low-order four-bit field propagated through all high-order bits, but it would not clear the carry flag otherwise.

If the carry flag is now set, or if the four high-order bits now exceed nine (1010xxxx-1111xxxx), these high-order bits are incremented by six, producing the proper BCD digit in the high-order nibble. Again, this would set the carry flag if there was a carry-out of the high-order bits, but wouldn't clear the carry. The carry flag thus indicates if the sum of the original two BCD variables is greater than 100, allowing multiple precision decimal addition. OV is not affected.

All of this occurs during the one instruction cycle. Essentially, this instruction performs the decimal conversion by adding 00H, 06H, 60H, or 66H to the accumulator, depending on initial accumulator and PSW conditions.

*Note: DA A cannot simply convert a hexadecimal number in the accumulator to BCD notation, nor does DA A apply to decimal subtraction.*

**Example:** The accumulator holds the value 56H (01010110B) representing the packed BCD digits of the decimal number 56. Register 3 contains the value 67H (01100111B) representing the packed BCD digits of the decimal number 67. The carry flag is set. The instruction sequence,

```
ADDC  A,R3
DA     A
```

will first perform a standard twos-complement binary addition, resulting in the value 0BEH (10111110) in the accumulator. The carry and auxiliary carry flags will be cleared.

The Decimal Adjust instruction will then alter the accumulator to the value 24H (00100100B), indicating the packed BCD digits of the decimal number 24, the low-order two digits of the decimal sum of 56, 67, and the carry-in. The carry flag will be set by the Decimal Adjust instruction, indicating that a decimal overflow occurred. The true sum 56, 67, and 1 is 124.

BCD variables can be incremented or decremented by adding 01H or 99H. If the accumulator initially holds 30H (representing the digits of 30 decimal), then the instruction sequence,

```
ADD  A,#99H
DA   A
```

will leave the carry set and 29H in the accumulator, since  $30 + 99 = 129$ . The low-order byte of the sum can be interpreted to mean  $30 - 1 = 29$ .

**Bytes:**  
**Cycles:**

1  
1

**Encoding:**

1	1	0	1	0	1	0	0
---	---	---	---	---	---	---	---

**Operation:**

```
DA
-contents of Accumulator are BCD
IF  [[(A3-0) > 9] ∨ [(AC) = 1]]
    THEN (A3-0) ← (A3-0) + 6
    AND
IF  [[(A7-4) > 9] ∨ [(C) = 1]]
    THEN (A7-4) ← (A7-4) + 6
```

**DEC byte****Function:** Decrement**Description:** The variable indicated is decremented by 1. An original value of 00H will underflow to 0FFH. No flags are affected. Four operand addressing modes are allowed: accumulator, register, direct, or register-indirect.*Note:* When this instruction is used to modify an output port, the value used as the original port data will be read from the output data latch, *not* the input pins.**Example:** Register 0 contains 7FH (01111111B). Internal RAM locations 7EH and 7FH contain 00H and 40H, respectively. The instruction sequence,

DEC @R0

DEC R0

DEC @R0

will leave register 0 set to 7EH and internal RAM locations 7EH and 7FH set to 0FFH and 3FH.

**DEC A****Bytes:** 1**Cycles:** 1**Encoding:**

0	0	0	1	0	1	0	0
---	---	---	---	---	---	---	---

**Operation:**

DEC

 $(A) \leftarrow (A) - 1$ **DEC Rn****Bytes:** 1**Cycles:** 1**Encoding:**

0	0	0	1	1	r	r	r
---	---	---	---	---	---	---	---

**Operation:**

DEC

 $(Rn) \leftarrow (Rn) - 1$

# DEC direct

Bytes: 2

Cycles: 1

Encoding: 0 0 0 1 0 1 0 1 direct address

Operation: DEC  
 $(\text{direct}) \leftarrow (\text{direct}) - 1$

# DEC @Ri

Bytes: 1

Cycles: 1

Encoding: 0 0 0 1 0 1 1 1

Operation: DEC  
 $((Ri)) \leftarrow ((Ri)) - 1$

# DIV AB

Function: Divide

Description: DIV AB divides the unsigned eight-bit integer in the accumulator by the unsigned eight-bit integer in register B. The accumulator receives the integer part of the quotient; register B receives the integer remainder. The carry and OV flags will be cleared.

Exception: if B had originally contained 00H, the values returned in the accumulator and B-register will be undefined and the overflow flag will be set. The carry flag is cleared in any case.

Example: The accumulator contains 251 (0FBH or 11111011B) and B contains 18 (12H or 00010010B). The instruction,

DIV AB

will leave 13 in the accumulator (0DH or 00001101B) and the value 17 (11H or 00010001B) in B, since  $251 = (13 \times 18) + 17$ . Carry and OV will both be cleared.

Bytes: 1

Cycles: 4

Encoding: 1 0 0 0 0 1 0 0

Operation: DIV  
 $(A)_{15-8} \leftarrow (A) / (B)$   
 $(B)_{7-0}$



**DJNZ** <byte>, <rel-addr>**Function:** Decrement and Jump if Not Zero**Description:** DJNZ decrements the location indicated by 1, and branches to the address indicated by the second operand if the resulting value is not zero. An original value of 00H will underflow to 0FFH. No flags are affected. The branch destination would be computed by adding the signed relative-displacement value in the last instruction byte to the PC, after incrementing the PC to the first byte of the following instruction.

The location decremented may be a register or directly addressed byte.

*Note:* When this instruction is used to modify an output port, the value used as the original port data will be read from the output data latch, *not* the input pins.**Example:** Internal RAM locations 40H, 50H, and 60H contain the values 01H, 70H, and 15H, respectively. The instruction sequence,

DJNZ 40H, LABEL\_1

DJNZ 50H, LABEL\_2

DJNZ 60H, LABEL\_3

will cause a jump to the instruction at label LABEL\_2 with the values 00H, 6FH, and 15H in the three RAM locations. The first jump was *not* taken because the result was zero.

This instruction provides a simple way of executing a program loop a given number of times, or for adding a moderate time delay (from 2 to 512 machine cycles) with a single instruction. The instruction sequence,

```

MOV R2, #8
TOGGLE: CPL P1.7
        DJNZ R2, TOGGLE

```

will toggle P1.7 eight times, causing four output pulses to appear at bit 7 of output port 1. Each pulse will last three machine cycles; two for DJNZ and one to alter the pin.

1	0	0	0	1	0	0	0
---	---	---	---	---	---	---	---

# DJNZ Rn,rel

Bytes: 2  
Cycles: 2

Encoding: 1 1 0 1 1 r r r direct address

Operation: DJNZ  
 $(PC) \leftarrow (PC) + 2$   
 $(Rn) \leftarrow (Rn) - 1$   
IF  $(Rn) > 0$  or  $(Rn) < 0$   
THEN  
 $(PC) \leftarrow (PC) + rel$

# DJNZ direct,rel

Bytes: 3  
Cycles: 2

Encoding: 1 1 0 1 0 1 0 1 direct address rel. address

Operation: DJNZ  
 $(PC) \leftarrow (PC) + 2$   
 $(direct) \leftarrow (direct) - 1$   
IF  $(direct) > 0$  or  $(direct) < 0$   
THEN  
 $(PC) \leftarrow (PC) + rel$

# INC <byte>

Function: Increment  
Description: INC increments the indicated variable by 1. An original value of 0FFH will overflow to 00H. No flags are affected. Three addressing modes are allowed: register, direct, or register-indirect.

Note: When this instruction is used to modify an output port, the value used as the original port data will be read from the output data latch, *not* the input pins.

**Example:** Register 0 contains 7EH (01111110B). Internal RAM locations 7EH and 7FH contain 0FFH and 40H, respectively. The instruction sequence,

INC @R0  
INC R0  
INC @R0

will leave register 0 set to 7FH and internal RAM locations 7EH and 7FH holding (respectively) 00H and 41H.

# INC A

Bytes: 1  
Cycles: 1

Encoding: 0 0 0 0 0 1 0 0

Operation: INC  
(A) ← (A) + 1

# INC Rn

Bytes: 1  
Cycles: 1

Encoding: 0 0 0 0 1 r r r

Operation: INC  
(Rn) ← (Rn) + 1

# INC direct

Bytes: 2  
Cycles: 1

Encoding: 0 0 0 0 0 1 0 1 direct address

Operation: INC  
(direct) ← (direct) + 1

<b>INC @Ri</b>		1B	bit,rel
<b>Bytes:</b>	1		
<b>Cycles:</b>	1		

**Encoding:**

0	0	0	0	0	1	1	i
---	---	---	---	---	---	---	---

**Operation:** INC  
 $((Ri)) \leftarrow ((Ri)) + 1$

**INC DPTR**

**Function:** Increment Data Pointer  
**Description:** Increment the 16-bit data pointer by 1. A 16-bit increment (modulo 2<sup>16</sup>) is performed; an overflow of the low-order byte of the data pointer (DPL) from 0FFH to 00H will increment the high-order byte (DPH). No flags are affected.

**Example:** This is the only 16-bit register which can be incremented.  
 Registers DPH and DPL contain 12H and 0FEH, respectively. The instruction sequence,

```
INC DPTR
INC DPTR
INC DPTR
```

will change DPH and DPL to 13H and 01H.

<b>Bytes:</b>	1
<b>Cycles:</b>	2

**Encoding:**

1	0	1	0	0	0	1	1
---	---	---	---	---	---	---	---

**Operation:** INC  
 $(DPTR) \leftarrow (DPTR) + 1$



**JB     bit,rel****Function:** Jump if Bit set**Description:** If the indicated bit is a one, jump to the address indicated; otherwise proceed with the next instruction. The branch destination is computed by adding the signed relative-displacement in the third instruction byte to the PC, after incrementing the PC to the first byte of the next instruction. *The bit tested is not modified.* No flags are affected.**Example:** The data present at input port 1 is 11001010B. The accumulator holds 56 (01010110B). The instruction sequence,

```
JB    P1.2,LABEL1
JB    ACC.2,LABEL2
```

will cause program execution to branch to the instruction at label LABEL2.

**Bytes:** 3**Cycles:** 2**Encoding:**

0 0 1 0 0 0 0 0

bit address

rel. address

**Operation:**

```
JB
(PC) ← (PC) + 3
IF (bit) = 1
    THEN
        (PC) ← (PC) + rel
```

**JBC     bit,rel****Function:** Jump if Bit is set and Clear bit**Description:** If the indicated bit is one, branch to the address indicated; otherwise proceed with the next instruction. *In either case, clear the designated bit.* The branch destination is computed by adding the signed relative-displacement in the third instruction byte to the PC, after incrementing the PC to the first byte of the next instruction. No flags are affected.*Note:* When this instruction is used to test an output pin, the value used as the original data will be read from the output data latch, *not* the input pin.**Example:** The accumulator holds 56H (01010110B). The instruction sequence,

```
JBC   ACC.3,LABEL1
JBC   ACC.2,LABEL2
```

will cause program execution to continue at the instruction identified by the label LABEL2, with the accumulator modified to 52H (01010010B).

Bytes: 3  
Cycles: 2

Encoding: 

0	0	0	1	0	0	0	0
---	---	---	---	---	---	---	---

bit address
-------------

rel. address
--------------

Operation: JBC  
 $(PC) \leftarrow (PC) + 3$   
 IF (bit) = 1  
 THEN  
 $(bit) \leftarrow 0$   
 $(PC) \leftarrow (PC) + rel$

JC rel

**Function:** Jump if Carry is set  
**Description:** If the carry flag is set, branch to the address indicated; otherwise proceed with the next instruction. The branch destination is computed by adding the signed relative-displacement in the second instruction byte to the PC, after incrementing the PC twice. No flags are affected.

**Example:** The carry flag is cleared. The instruction sequence,

```
JC LABEL1
CPL C
JC LABEL2
```

will set the carry and cause program execution to continue at the instruction identified by the label LABEL2.

Bytes: 2  
Cycles: 2

Encoding: 

0	1	0	0	0	0	0	0
---	---	---	---	---	---	---	---

rel. address
--------------

Operation: JC  
 $(PC) \leftarrow (PC) + 2$   
 IF (C) = 1  
 THEN  
 $(PC) \leftarrow (PC) + rel$

**JMP    @A + DPTR****Function:** Jump indirect**Description:** Add the eight-bit unsigned contents of the accumulator with the sixteen-bit data pointer, and load the resulting sum to the program counter. This will be the address for subsequent instruction fetches. Sixteen-bit addition is performed (modulo 216): a carry-out from the low-order eight bits propagates through the higher-order bits. Neither the accumulator nor the data pointer is altered. No flags are affected.**Example:** An even number from 0 to 6 is in the accumulator. The following sequence of instructions will branch to one of four AJMP instructions in a jump table starting at JMP\_\_TBL:

```

                MOV     DPTR,#JMP__TBL
                JMP     @A + DPTR
JMP__TBL:  AJMP    LABEL0
            AJMP    LABEL1
            AJMP    LABEL2
            AJMP    LABEL3

```

If the accumulator equals 04H when starting this sequence, execution will jump to label LABEL2. Remember that AJMP is a two-byte instruction, so the jump instructions start at every other address.

**Bytes:** 1**Cycles:** 2**Encoding:**

0	1	1	1	0	0	1	1
---	---	---	---	---	---	---	---

**Operation:**

JMP  
 $(PC) \leftarrow (A) + (DPTR)$

**JNB** bit,rel**Function:** Jump if Bit Not set**Description:** If the indicated bit is a zero, branch to the indicated address; otherwise proceed with the next instruction. The branch destination is computed by adding the signed relative-displacement in the third instruction byte to the PC, after incrementing the PC to the first byte of the next instruction. *The bit tested is not modified.* No flags are affected.**Example:** The data present at input port 1 is 11001010B. The accumulator holds 56H (01010110B). The instruction sequence,

```
JNB P1.3,LABEL1
JNB ACC.3,LABEL2
```

will cause program execution to continue at the instruction at label LABEL2.

**Bytes:** 3**Cycles:** 2**Encoding:** 0 0 1 1 0 0 0 0 bit address rel. address

**Operation:** JNB  
 $(PC) \leftarrow (PC) + 3$   
 IF (bit) = 0  
 THEN  $(PC) \leftarrow (PC) + \text{rel.}$

**JNC** rel**Function:** Jump if Carry not set**Description:** If the carry flag is a zero, branch to the address indicated; otherwise proceed with the next instruction. The branch destination is computed by adding the signed relative-displacement in the second instruction byte to the PC, after incrementing the PC twice to point to the next instruction. The carry flag is not modified.**Example:** The carry flag is set. The instruction sequence,

```
JNC LABEL1
CPL C
JNC LABEL2
```

will clear the carry and cause program execution to continue at the instruction identified by the label LABEL2.



Bytes: 2

Cycles: 2

Encoding: 0 1 0 1 0 0 0 0 rel. address

**Operation:** JNC  
 $(PC) \leftarrow (PC) + 2$   
 IF  $(C) = 0$   
 THEN  $(PC) \leftarrow (PC) + \text{rel}$

JNZ rel

**Function:** Jump if accumulator Not Zero

**Description:** If any bit of the accumulator is a one, branch to the indicated address; otherwise proceed with the next instruction. The branch destination is computed by adding the signed relative-displacement in the second instruction byte to the PC, after incrementing the PC twice. The accumulator is not modified. No flags are affected.

**Example:** The accumulator originally holds 00H. The instruction sequence,

```
JNZ LABEL1
INC A
JNZ LABEL2
```

will set the accumulator to 01H and continue at label LABEL2.

Bytes: 2

Cycles: 2

Encoding: 0 1 1 1 0 0 0 0 rel. address

**Operation:** JNZ  
 $(PC) \leftarrow (PC) + 2$   
 IF  $(A) \neq 0$   
 THEN  $(PC) \leftarrow (PC) + \text{rel}$

**JZ rel**

**Function:** Jump if Accumulator Zero

**Description:** If all bits of the accumulator are zero, branch to the address indicated; otherwise proceed with the next instruction. The branch destination is computed by adding the signed relative-displacement in the second instruction byte to the PC, after incrementing the PC twice. The accumulator is not modified. No flags are affected.

**Example:** The accumulator originally contains 01H. The instruction sequence,

```
JZ LABEL1  
DEC A  
JZ LABEL2
```

will change the accumulator to 00H and cause program execution to continue at the instruction identified by the label LABEL2.

**Bytes:** 2

**Cycles:** 2

**Encoding:**

0	1	1	0	0	0	0	0
---	---	---	---	---	---	---	---

rel. address

**Operation:**

JZ

$(PC) \leftarrow (PC) + 2$

IF  $(A) = 0$

THEN  $(PC) \leftarrow (PC) + \text{rel}$

**LCALL addr16**

**Function:** Long Call

**Description:** LCALL calls a subroutine located at the indicated address. The instruction adds three to the program counter to generate the address of the next instruction and then pushes the 16-bit result onto the stack (low byte first), incrementing the stack pointer by two. The high-order and low-order bytes of the PC are then loaded, respectively, with the second and third bytes of the LCALL instruction. Program execution continues with the instruction at this address. The subroutine may therefore begin anywhere in the full 64K-byte program memory address space. No flags are affected.

**Example:** Initially the stack pointer equals 07H. The label "SUBRTN" is assigned to program memory location 1234H. After executing the instruction,

**LCALL SUBRTN**

at location 0123H, the stack pointer will contain 09H, internal RAM locations 08H and 09H will contain 26H and 01H, and the PC will contain 1235H.

**Bytes:** 3

**Cycles:** 2

**Encoding:**

0 0 0 1 0 0 1 0

addr15 - addr8

addr7 - addr0

**Operation:**

**LCALL**

$(PC) \leftarrow (PC) + 3$

$(SP) \leftarrow (SP) + 1$

$((SP)) \leftarrow (PC_{7-0})$

$(SP) \leftarrow (SP) + 1$

$((SP)) \leftarrow (PC_{15-8})$

$(PC) \leftarrow \text{addr}_{15-0}$

**LJMP addr16**

**Function:** Long Jump

**Description:** LJMP causes an unconditional branch to the indicated address, by loading the high-order and low-order bytes of the PC (respectively) with the second and third instruction bytes. The destination may therefore be anywhere in the full 64K program memory address space. No flags are affected.

**Example:** The label "JMPADR" is assigned to the instruction at program memory location 1234H. The instruction,

**LJMP JMPADR**

at location 0123H will load the program counter with 1234H.

**Bytes:** 3

**Cycles:** 2

**Encoding:**

0 0 0 0 0 0 1 0

addr15 - addr8

addr7 - addr0

**Operation:**

**LJMP**

$(PC) \leftarrow \text{addr}_{15-0}$

# MCS®-51 INSTRUCTION SET

## MOV <dest-byte>,<src-byte>

**Function:** Move byte variable  
**Description:** The byte variable indicated by the second operand is copied into the location specified by the first operand. The source byte is not affected. No other register or flag is affected.

**Example:** This is by far the most flexible operation. Fifteen combinations of source and destination addressing modes are allowed.  
 Internal RAM location 30H holds 40H. The value of RAM location 40H is 10H. The data present at input port 1 is 11001010B (0CAH).

```
MOV R0,#30H      ;R0<= 30H
MOV A,@R0        ;A <= 40H
MOV R1,A         ;R1 <= 40H
MOV R,@R1        ;B <= 10H
MOV @R1,P1       ;RAM (40H) <= 0CAH
MOV P2,P1        ;P2 #0CAH
```

leaves the value 30H in register 0, 40H in both the accumulator and register 1, 10H in register B, and 0CAH (11001010B) both in RAM location 40H and output on port 2.

### MOV A,Rn

**Bytes:** 1  
**Cycles:** 1

**Encoding:** 1 1 1 0 1 r r r

**Operation:** MOV  
 (A) ← (Rn)

### \*MOV A,direct

**Bytes:** 2  
**Cycles:** 1

**Encoding:** 1 1 1 0 0 1 0 1 direct address

**Operation:** MOV  
 (A) ← (direct)

\*MOV A,ACC is not a valid instruction.



# MCS®-51 INSTRUCTION SET

**MOV   A,@Ri**

**Bytes:** 1

**Cycles:** 1

**Encoding:**

1	1	1	0	0	1	1	i
---	---	---	---	---	---	---	---

**Operation:** MOV

$$(A) \longleftarrow ((Ri))$$

**MOV A,#data**

**Bytes:** 2

**Cycles:** 1

**Encoding:**

0	1	1	1	0	1	0	0
---	---	---	---	---	---	---	---

immediate data

**Operation:** MOV

(A) ← #data

**MOV Rn,A**

**Bytes:** 1

**Cycles:** 1

**Encoding:**

1	1	1	1	1	r	r	r
---	---	---	---	---	---	---	---

**Operation:** MOV

$$(R_n) \leftarrow (A)$$

## MOV Rn,direct

**Bytes:** 2

**Cycles:** 2

**Encoding:**

1	0	1	0	1	r	r	r
---	---	---	---	---	---	---	---

direct addr.

**Operation:** MOV

(Rn) ← (direct)

### MOV Rn,#data

**Bytes:** 2

Cycles: 1

**Encoding:**

0	1	1	1	1	r	r	r
---	---	---	---	---	---	---	---

immediate data

**Operation:** MOV

$$(R_n) \leftarrow \#data$$

# MCS®-51 INSTRUCTION SET

**MOV direct,A**

Bytes: 2

Cycles: 1

Encoding:

1 1 1 1 0 1 0 1

direct address

Operation:

MOV

(direct) ← (A)

**MOV direct,Rn**

Bytes: 2

Cycles: 2

Encoding:

1 0 0 0 1 r r r

direct address

Operation:

MOV

(direct) ← (Rn)

**MOV direct,direct**

Bytes: 3

Cycles: 2

Encoding:

1 0 0 0 0 1 0 1

dir. addr. (src)

dir. addr. (dest)

Operation:

MOV

(direct) ← (direct)

**MOV direct,@Ri**

Bytes: 2

Cycles: 2

Encoding:

1 0 0 0 0 1 1 i

direct addr.

Operation:

MOV

(direct) ← ((Ri))

**MOV direct,#data**

Bytes: 3

Cycles: 2

Encoding:

0 1 1 1 0 1 0 1

direct address

immediate data

Operation:

MOV

(direct) ← #data

**MOV @Ri,A**

**Bytes:** 1

**Cycles:** 1

**Encoding:**

1 1 1 1 0 1 1 i

**Operation:**

MOV  
((Ri)) ← (A)

**MOV @Ri,direct**

**Bytes:** 2

**Cycles:** 2

**Encoding:**

1 0 1 0 0 1 1 i

direct addr.

0 0 0 1

**Operation:**

MOV  
((Ri)) ← (direct)

**MOV @Ri,#data**

**Bytes:** 2

**Cycles:** 1

**Encoding:**

0 1 1 1 0 1 1 i

immediate data

0 0 1

**Operation:**

MOV  
((Ri)) ← #data

**MOV <dest-bit>,<src-bit>**

**Function:**

Move bit data

**Description:**

The Boolean variable indicated by the second operand is copied into the location specified by the first operand. One of the operands must be the carry flag; the other may be any directly addressable bit. No other register or flag is affected.

**Example:**

The carry flag is originally set. The data present at input port 3 is 11000101B. The data previously written to output port 1 is 35H (00110101B).

MOV P1.3,C

MOV C,P3.3

MOV P1.2,C

will leave the carry cleared and change port 1 to 39H (00111001B).

**MOV C,bit****Bytes:** 2**Cycles:** 1**Encoding:**

1 0 1 0 0 0 1 0

bit address

**Operation:**

MOV

(C) ← (bit)

**MOV bit,C****Bytes:** 2**Cycles:** 2**Encoding:**

1 0 0 1 0 0 1 0

bit address

**Operation:**

MOV

(bit) ← (C)

**MOV DPTR,#data16****Function:**

Load Data Pointer with a 16-bit constant

**Description:**

The data pointer is loaded with the 16-bit constant indicated. The 16-bit constant is loaded into the second and third bytes of the instruction. The second byte (DPH) is the high-order byte, while the third byte (DPL) holds the low-order byte. No flags are affected.

**Example:**

This is the only instruction which moves 16 bits of data at once.

The instruction,

MOV DPTR,#1234H

will load the value 1234H into the data pointer: DPH will hold 12H and DPL will hold 34H.

**Bytes:** 3**Cycles:** 2**Encoding:**

1 0 0 1 0 0 0 0

immed. data15 - 8

immed. data7 - 0

**Operation:**

MOV

(DPTR) ← #data15-0

DPH □ DPL ← #data15-8 □ #data7-0



**MOVC    A,@A + <base-reg>**

**Function:** Move Code byte

**Description:** The MOVC instructions load the accumulator with a code byte, or constant from program memory. The address of the byte fetched is the sum of the original unsigned eight-bit accumulator contents and the contents of a sixteen-bit base register, which may be either the data pointer or the PC. In the latter case, the PC is incremented to the address of the following instruction before being added with the accumulator; otherwise the base register is not altered. Sixteen-bit addition is performed so a carry-out from the low-order eight bits may propagate through higher-order bits. No flags are affected.

**Example:** A value between 0 and 3 is in the accumulator. The following instructions will translate the value in the accumulator to one of four values defined by the DB (define byte) directive.

```
REL_PC: INC     A
          MOVC   A,@A + PC
          RET
          DB      66H
          DB      77H
          DB      88H
          DB      99H
```

If the subroutine is called with the accumulator equal to 01H, it will return with 77H in the accumulator. The INC A before the MOVC instruction is needed to "get around" the RET instruction above the table. If several bytes of code separated the MOVC from the table, the corresponding number would be added to the accumulator instead.

**MOVC    A,@A + DPTR****Bytes:** 1**Cycles:** 2**Encoding:**

1	0	0	1	0	0	1	1
---	---	---	---	---	---	---	---

**Operation:** MOVC  
(A) ← ((A) + (DPTR))

**MOVC A,@A+PC****Bytes:** 1**Cycles:** 2**Encoding:**

1	0	0	0	0	0	1	1
---	---	---	---	---	---	---	---

**Operation:** MOVC $(PC) \leftarrow (PC) + 1$  $(A) \leftarrow ((A) + (PC))$ **MOVX <dest-byte>,<src-byte>****Function:** Move External**Description:** The MOVX instructions transfer data between the accumulator and a byte of external data memory, hence the "X" appended to MOV. There are two types of instructions, differing in whether they provide an eight-bit or sixteen-bit indirect address to the external data RAM.

In the first type, the contents of R0 or R1 in the current register bank provide an eight-bit address multiplexed with data on P0. Eight bits are sufficient for external I/O expansion decoding or a relatively small RAM array. For somewhat larger arrays, any output port pins can be used to output higher-order address bits. These pins would be controlled by an output instruction preceding the MOVX.

In the second type of MOVX instruction, the data pointer generates a sixteen-bit address. P2 outputs the high-order eight address bits (the contents of DPH) while P0 multiplexes the low-order eight bits (DPL) with data. The P2 Special Function Register retains its previous contents while the P2 output buffers are emitting the contents of DPH. This form is faster and more efficient when accessing very large data arrays (up to 64K bytes), since no additional instructions are needed to set up the output ports.

It is possible in some situations to mix the two MOVX types. A large RAM array with its high-order address lines driven by P2 can be addressed via the data pointer, or with code to output high-order address bits to P2 followed by a MOVX instruction using R0 or R1.

**Example:** An external 256 byte RAM using multiplexed address/data lines (e.g., an Intel® 8155 RAM/I/O/Timer) is connected to the 8051 Port 0. Port 3 provides control lines for the external RAM. Ports 1 and 2 are used for normal I/O. Registers 0 and 1 contain 12H and 34H. Location 34H of the external RAM holds the value 56H. The instruction sequence,

```
MOVX A,@R1
MOVX @R0,A
```

copies the value 56H into both the accumulator and external RAM location 12H.

#### MOVX A,@Ri

Bytes: 1  
Cycles: 2

Encoding:

1	1	1	0	0	0	1	i
---	---	---	---	---	---	---	---

Operation: MOVX  
(A) ← ((Ri))

#### MOVX A,@DPTR

Bytes: 1  
Cycles: 2

Encoding:

1	1	1	0	0	0	0	0
---	---	---	---	---	---	---	---

Operation: MOVX  
(A) ← ((DPTR))

#### MOVX @Ri,A

Bytes: 1  
Cycles: 2

Encoding:

1	1	1	1	0	0	1	i
---	---	---	---	---	---	---	---

Operation: MOVX  
((Ri)) ← (A)

#### MOVX @DPTR,A

Bytes: 1  
Cycles: 2

Encoding:

1	1	1	1	0	0	0	0
---	---	---	---	---	---	---	---

Operation: MOVX  
(DPTR) ← (A)

**MUL AB****Function:** Multiply

**Description:** MUL AB multiplies the unsigned eight-bit integers in the accumulator and register B. The low-order byte of the sixteen-bit product is left in the accumulator, and the high-order byte in B. If the product is greater than 255 (0FFH) the overflow flag is set; otherwise it is cleared. The carry flag is always cleared.

**Example:** Originally the accumulator holds the value 80 (50H). Register B holds the value 160 (0A0H). The instruction,

MUL AB

will give the product 12,800 (3200H), so B is changed to 32H (00110010B) and the accumulator is cleared. The overflow flag is set, carry is cleared.

**Bytes:** 1

**Cycles:** 4

**Encoding:**

1	0	1	0	0	1	0	0
---	---	---	---	---	---	---	---

**Operation:**

MUL

(A)7-0 ← (A) X (B)

(B)15-8



NOP

<b>Function:</b>	No Operation										
<b>Description:</b>	Execution continues at the following instruction. Other than the PC, no registers or flags are affected.										
<b>Example:</b>	It is desired to produce a low-going output pulse on bit 7 of port 2 lasting exactly 5 cycles. A simple SETB/CLR sequence would generate a one-cycle pulse, so four additional cycles must be inserted. This may be done (assuming no interrupts are enabled) with the instruction sequence,										
	CLR	P2.7									
	NOP										
	NOP										
	NOP										
	NOP										
	SETB	P2.7									
<b>Bytes:</b>	1										
<b>Cycles:</b>	1										
<b>Encoding:</b>	<table border="1"><tr><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td></tr></table>			0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0				
<b>Operation:</b>	NOP $(PC) \leftarrow (PC) + 1$										

**ORL**    <dest-byte> <src-byte>**Function:** Logical-OR for byte variables**Description:** ORL performs the bitwise logical-OR operation between the indicated variables, storing the results in the destination byte. No flags are affected.

The two operands allow six addressing mode combinations. When the destination is the accumulator, the source can use register, direct, register-indirect, or immediate addressing; when the destination is a direct address, the source can be the accumulator or immediate data.

*Note:* When this instruction is used to modify an output port, the value used as the original port data will be read from the output data latch, *not* the input pins.

**Example:** If the accumulator holds 0C3H (11000011B) and R0 holds 55H (01010101B) then the instruction,

```
ORL    A,R0
```

will leave the accumulator holding the value 0D7H (11010111B).

When the destination is a directly addressed byte, the instruction can set combinations of bits in any RAM location or hardware register. The pattern of bits to be set is determined by a mask byte, which may be either a constant data value in the instruction or a variable computed in the accumulator at run-time. The instruction,

```
ORL    P1,#00110010B
```

will set bits 5, 4, and 1 of output port 1.

**ORL A,Rn****Bytes:** 1**Cycles:** 1**Encoding:**

0	1	0	0	1	r	r	r
---	---	---	---	---	---	---	---

**Operation:**

ORL

 $(A) \leftarrow (A) \vee (Rn)$

**ORL A,direct**

Bytes: 2

Cycles: 1

Encoding: 

0	1	0	0
---	---	---	---

0	1	0	1
---	---	---	---

direct address			
----------------	--	--	--

Operation: ORL  
 $(A) \leftarrow (A) \vee (\text{direct})$

**ORL A,@Ri**

Bytes: 1

Cycles: 1

Encoding: 

0	1	0	0
---	---	---	---

0	1	1	i
---	---	---	---

Operation: ORL  
 $(A) \leftarrow (A) \vee ((Ri))$

**ORL A,#data**

Bytes: 2

Cycles: 1

Encoding: 

0	1	0	0
---	---	---	---

0	1	0	0
---	---	---	---

immediate data			
----------------	--	--	--

Operation: ORL  
 $(A) \leftarrow (A) \vee \#data$

**ORL direct,A**

Bytes: 2

Cycles: 1

Encoding: 

0	1	0	0
---	---	---	---

0	0	1	0
---	---	---	---

direct address			
----------------	--	--	--

Operation: ORL  
 $(\text{direct}) \leftarrow (\text{direct}) \vee (A)$

**ORL direct,#data**

Bytes: 3

Cycles: 2

Encoding: 

0	1	0	0
---	---	---	---

0	0	1	1
---	---	---	---

direct addr.			
--------------	--	--	--

immediate data			
----------------	--	--	--

Operation: ORL  
 $(\text{direct}) \leftarrow (\text{direct}) \vee \#data$

**ORL C, <src-bit>****Function:** Logical-OR for bit variables**Description:** Set the carry flag if the Boolean value is a logical 1; leave the carry in its current state otherwise. A slash ("/") preceding the operand in the assembly language indicates that the logical complement of the addressed bit is used as the source value, but the source bit itself is not affected. No other flags are affected.**Example:** Set the carry flag if and only if P1.0 = 1, ACC.7 = 1, or OV = 0:

```

MOV C,P1.0          ;LOAD CARRY WITH INPUT PIN P10
ORL C,ACC.7          ;OR CARRY WITH THE ACC. BIT 7
ORL C,/OV            ;OR CARRY WITH THE INVERSE OF OV

```

**ORL C,bit****Bytes:** 2**Cycles:** 2**Encoding:**

0 1 1 1 0 0 1 0

bit address

**Operation:**

ORL  
 $(C) \leftarrow (C) \vee (\text{bit})$

**ORL C,/bit****Bytes:** 2**Cycles:** 2**Encoding:**

1 0 1 0 0 0 0 0

bit address

**Operation:**

ORL  
 $(C) \leftarrow (C) \vee (\overline{\text{bit}})$

**POP direct****Function:** Pop from stack.**Description:** The contents of the internal RAM location addressed by the stack pointer is read, and the stack pointer is decremented by one. The value read is the transfer to the directly addressed byte indicated. No flags are affected.

direct address

1 1 0 0 0 0 0 0



**Example:** The stack pointer originally contains the value 32H, and internal RAM locations 30H through 32H contain the values 20H, 23H, and 01H, respectively. The instruction sequence,

```
POP    DPH
POP    DPL
```

will leave the stack pointer equal to the value 30H and the data pointer set to 0123H. At this point the instruction,

```
POP    SP
```

will leave the stack pointer set to 20H. Note that in this special case the stack pointer was decremented to 2FH before being loaded with the value popped (20H).

**Bytes:** 2  
**Cycles:** 2

**Encoding:**

1	1	0	1	0	0	0	0
---	---	---	---	---	---	---	---

direct address

**Operation:**

POP

(direct) ← ((SP))

(SP) ← (SP) - 1

**PUSH      direct**

**Function:**

Push onto stack

**Description:**

The stack pointer is incremented by one. The contents of the indicated variable is then copied into the internal RAM location addressed by the stack pointer. Otherwise no flags are affected.

**Example:**

On entering an interrupt routine the stack pointer contains 09H. The data pointer holds the value 0123H. The instruction sequence,

```
PUSH    DPL
PUSH    DPH
```

will leave the stack pointer set to 0BH and store 23H and 01H in internal RAM locations 0AH and 0BH, respectively.

**Bytes:** 2  
**Cycles:** 2

**Encoding:**

1	1	0	0	0	0	0	0
---	---	---	---	---	---	---	---

direct address

**Operation:**

PUSH

(SP) ← (SP) + 1

((SP)) ← (direct)

## RET

<b>Function:</b>	Return from subroutine								
<b>Description:</b>	RET pops the high- and low-order bytes of the PC successively from the stack, decrementing the stack pointer by two. Program execution continues at the resulting address, generally the instruction immediately following an ACALL or LCALL. No flags are affected.								
<b>Example:</b>	The stack pointer originally contains the value 0BH. Internal RAM locations 0AH and 0BH contain the values 23H and 01H, respectively. The instruction, RET  will leave the stack pointer equal to the value 09H. Program execution will continue at location 0123H.								
<b>Bytes:</b>	1								
<b>Cycles:</b>	2								
<b>Encoding:</b>	<table border="1"><tr><td>0</td><td>0</td><td>1</td><td>0</td><td>0</td><td>0</td><td>1</td><td>0</td></tr></table>	0	0	1	0	0	0	1	0
0	0	1	0	0	0	1	0		
<b>Operation:</b>	RET $(PC_{15-8}) \leftarrow ((SP))$ $(SP) \leftarrow (SP) - 1$ $(PC_{7-0}) \leftarrow ((SP))$ $(SP) \leftarrow (SP) - 1$								

## RETI

<b>Function:</b>	Return from interrupt
<b>Description:</b>	RETI pops the high- and low-order bytes of the PC successively from the stack, and restores the interrupt logic to accept additional interrupts at the same priority level as the one just processed. The stack pointer is left decremented by two. No other registers are affected; the PSW is <i>not</i> automatically restored to its pre-interrupt status. Program execution continues at the resulting address, which is generally the instruction immediately after the point at which the interrupt request was detected. If a lower- or same-level interrupt had been pending when the RETI instruction is executed, that one instruction will be executed before the pending interrupt is processed.

**Example:** The stack pointer originally contains the value 0BH. An interrupt was detected during the instruction ending at location 0122H. Internal RAM locations 0AH and 0BH contain the values 23H and 01H, respectively. The instruction, RETI, will leave the stack pointer equal to 09H and return program execution to location 0123H.

**Bytes:** 1  
**Cycles:** 2

**Encoding:**

0	0	1	1	0	0	1	0
---	---	---	---	---	---	---	---

**Operation:** RETI  
(PC15-8) ← ((SP))  
(SP) ← (SP) - 1  
(PC7-0) ← ((SP))  
(SP) ← (SP) - 1

0	0	1	0	0	1	0	0
---	---	---	---	---	---	---	---

RETI  
(SP) ← (SP) - 1  
(PC7-0) ← ((SP))  
(SP) ← (SP) - 1  
(PC15-8) ← ((SP))

**Function:** Return from interrupt.  
**Description:** RETI pops the high- and low-order bytes of the PC successively from the stack, and restores the interrupt logic to accept additional interrupts at the same priority level as the one just processed. The stack pointer is left decremented by two. No other registers are affected; the PSW is not automatically restored to its pre-interrupt status. Program execution continues at the resulting address, which is generally the instruction immediately after the point at which the interrupt request was detected. If a lower- or same-level interrupt had been pending when the RETI instruction is executed, that one instruction will be executed before the pending interrupt is processed.

**RL    A****Function:** Rotate accumulator Left**Description:** The eight bits in the accumulator are rotated one bit to the left. Bit 7 is rotated into the bit 0 position. No flags are affected.**Example:** The accumulator holds the value 0C5H (11000101B). The instruction,

RL    A

leaves the accumulator holding the value 8BH (10001011B) with the carry unaffected.

**Bytes:** 1**Cycles:** 1**Encoding:**

0	0	1	0	0	0	1	1
---	---	---	---	---	---	---	---

**Operation:**

RL

 $(A_{n+1}) \leftarrow (A_n) \quad n=0-6$  $(A_0) \leftarrow (A_7)$ **RLC    A****Function:** Rotate accumulator Left through the Carry flag**Description:** The eight bits in the accumulator and the carry flag are together rotated one bit to the left. Bit 7 moves into the carry flag; the original state of the carry flag moves into the bit 0 position. No other flags are affected.**Example:** The accumulator holds the value 0C5H (11000101B), and the carry is zero. The instruction,

RLC    A

leaves the accumulator holding the value 8BH (10001010B) with the carry set.

**Bytes:** 1**Cycles:** 1**Encoding:**

0	0	1	1	0	0	1	1
---	---	---	---	---	---	---	---

**Operation:**

RLC

 $(A_{n+1}) \leftarrow (A_n) \quad n=0-6$  $(A_0) \leftarrow (C)$  $(C) \leftarrow (A_7)$



**RR A****Function:** Rotate accumulator Right**Description:** The eight bits in the accumulator are rotated one bit to the right. Bit 0 is rotated into the bit 7 position. No flags are affected.**Example:** The accumulator holds the value 0C5H (11000101B). The instruction,

RR A

leaves the accumulator holding the value 0E2H (11100010B) with the carry unaffected.

**Bytes:** 1**Cycles:** 1**Encoding:**

0	0	0	0	0	0	1	1
---	---	---	---	---	---	---	---

**Operation:**

RR

 $(A_n) \leftarrow (A_{n+1}) \quad n=0-6$  $(A_7) \leftarrow (A_0)$ **RRC A****Function:** Rotate accumulator Right through Carry flag**Description:** The eight bits in the accumulator and the carry flag are together rotated one bit to the right. Bit 0 moves into the carry flag; the original value of the carry flag moves into the bit 7 position. No other flags are affected.**Example:** The accumulator holds the value 0C5H (11000101B), the carry is zero. The instruction,

RRC A

leaves the accumulator holding the value 62 (01100010B) with the carry set.

**Bytes:** 1**Cycles:** 1**Encoding:**

0	0	0	1	0	0	1	1
---	---	---	---	---	---	---	---

**Operation:**

RRC

 $(A_n) \leftarrow (A_{n+1}) \quad n=0-6$  $(A_7) \leftarrow (C)$  $(C) \leftarrow (A_0)$

**SETB** <bit>**Function:** Set Bit**Description:** SETB sets the indicated bit to one. SETB can operate on the carry flag or any directly addressable bit. No other flags are affected.**Example:** The carry flag is cleared. Output port 1 has been written with the value 34H (00110100B). The instructions,

```

SETB C
SETB P1.0

```

will leave the carry flag set to 1 and change the data output on port 1 to 35H (00110101B).

**SETB C****Bytes:** 1**Cycles:** 1**Encoding:**

1	1	0	1	0	0	1	1
---	---	---	---	---	---	---	---

**Operation:**

SETB  
(C) ← 1

**SETB bit****Bytes:** 2**Cycles:** 1**Encoding:**

1	1	0	1	0	0	1	0
---	---	---	---	---	---	---	---

bit address

**Operation:**

SETB  
(bit) ← 1

**SJMP** rel**Function:** Short Jump**Description:** Program control branches unconditionally to the address indicated. The branch destination is computed by adding the signed displacement in the second instruction byte to the PC, after incrementing the PC twice. Therefore, the range of destinations allowed is from 128 bytes preceding this instruction to 127 bytes following it.

**Example:** The label "RELADR" is assigned to an instruction at program memory location 0123H. The instruction,

**SJMP RELADR**

will assemble into location 0100H. After the instruction is executed, the PC will contain the value 0123H.

(Note: Under the above conditions the instruction following SJMP will be at 102H. Therefore, the displacement byte of the instruction will be the relative offset  $(0123H - 0102H) = 21H$ . Put another way, an SJMP with a displacement of 0FEH would be a one-instruction infinite loop.)

**Bytes:** 2

**Cycles:** 2

**Encoding:**

1	0	0	0	0	0	0	0
---	---	---	---	---	---	---	---

rel. address

**Operation:**

**SJMP**

$(PC) \leftarrow (PC) + 2$

$(PC) \leftarrow (PC) + \text{rel}$

**SUBB A, <src-byte>**

**Function:** Subtract with borrow

**Description:** SUBB subtracts the indicated variable and the carry flag together from the accumulator, leaving the result in the accumulator. SUBB sets the carry (borrow) flag if a borrow is needed for bit 7, and clears C otherwise. (If C was set *before* executing a SUBB instruction, this indicates that a borrow was needed for the previous step in a multiple precision subtraction, so the carry is subtracted from the accumulator along with the source operand.) AC is set if a borrow is needed for bit 3, and cleared otherwise. OV is set if a borrow is needed into bit 6, but not into bit 7, or into bit 7, but not bit 6.

When subtracting signed integers OV indicates a negative number produced when a negative value is subtracted from a positive value, or a positive result when a positive number is subtracted from a negative number.

The source operand allows four addressing modes: register, direct, register-indirect, or immediate.

**Example:** The accumulator holds 0C9H (11001001B), register 2 holds 54H (01010100B), and the carry flag is set. The instruction,

SUBB A,R2

will leave the value 74H (01110100B) in the accumulator, with the carry flag and AC cleared but OV set.

Notice that 0C9H minus 54H is 75H. The difference between this and the above result is due to the carry (borrow) flag being set before the operation. If the state of the carry is not known before starting a single or multiple-precision subtraction, it should be explicitly cleared by a CLR C instruction.

**SUBB A,Rn**

**Bytes:** 1

**Cycles:** 1

**Encoding:** 1 0 0 1 1 r r r

**Operation:** SUBB  
 $(A) \leftarrow (A) - (C) - (Rn)$



**SUBB A,direct****Bytes:** 2**Cycles:** 1**Encoding:**

1 0 0 1 0 1 0 1

direct address

**Operation:**

SUBB

 $(A) \leftarrow (A) - (C) - (\text{direct})$ **SUBB A,@Ri****Bytes:** 1**Cycles:** 1**Encoding:**

1 0 0 1 0 1 1 i

**Operation:**

SUBB

 $(A) \leftarrow (A) - (C) - ((Ri))$ **SUBB A,#data****Bytes:** 2**Cycles:** 1**Encoding:**

1 0 0 1 0 1 0 0

immediate data

**Operation:**

SUBB

 $(A) \leftarrow (A) - (C) - \#data$ **SWAP A****Function:**

Swap nibbles within the Accumulator

**Description:**

SWAP A interchanges the low- and high-order nibbles (four-bit fields) of the accumulator (bits 3-0 and bits 7-4). The operation can also be thought of as a four-bit rotate instruction. No flags are affected.

**Example:**

The accumulator holds the value 0C5H (11000101B). The instruction,

SWAP A

leaves the accumulator holding the value 5CH (01011100B).

**Bytes:** 1**Cycles:** 1**Encoding:**

1 1 0 0 0 1 0 0

**Operation:**

SWAP

 $(A_{3-0}) \leftrightarrow (A_{7-4}), (A_{7-4}) \leftarrow (A_{3-0})$

**XCH A,<byte>****Function:** Exchange Accumulator with byte variable**Description:** XCH loads the accumulator with the contents of the indicated variable, at the same time writing the original accumulator contents to the indicated variable. The source/destination operand can use register, direct, or register-indirect addressing.**Example:** R0 contains the address 20H. The accumulator holds the value 3FH (00111111B). Internal RAM location 20H holds the value 75H (01101010B). The instruction,

XCH A,@R0

will leave RAM location 20H holding the values 3FH (00111111B) and 75H (01101010B) in the accumulator.

**XCH A,Rn****Bytes:** 1**Cycles:** 1**Encoding:**

1	1	0	0	1	r	r	r
---	---	---	---	---	---	---	---

**Operation:**
XCH  
 $(A) \longleftrightarrow (Rn)$ 
**XCH A,direct****Bytes:** 2**Cycles:** 1**Encoding:**

1	1	0	0	0	1	0	1
---	---	---	---	---	---	---	---

direct address

**Operation:**
XCH  
 $(A) \longleftrightarrow (\text{direct})$ 
**XCH A,@Ri****Bytes:** 1**Cycles:** 1**Encoding:**

1	1	0	0	0	1	1	i
---	---	---	---	---	---	---	---

**Operation:**
XCH  
 $(A) \longleftrightarrow ((Ri))$

XCHD    A,@Ri

**Function:** Exchange Digit**Description:** XCHD exchanges the low-order nibble of the accumulator (bits 3-0), generally representing a hexadecimal or BCD digit), with that of the internal RAM location indirectly addressed by the specified register. The high-order nibbles (bits 7-4) of each register are not affected. No flags are affected.**Example:** R0 contains the address 20H. The accumulator holds the value 36H (00110110B). Internal RAM location 20H holds the value 75H (01110101B). The instruction,

XCHD    A,@R0

will leave RAM location 20H holding the value 76H (01110110B) and 35H (00110101B) in the accumulator.

**Bytes:** 1**Cycles:** 1**Encoding:**

1	1	0	1	0	1	1	i
---	---	---	---	---	---	---	---

**Operation:**
XCHD  
 $(A_{3-0}) \longleftrightarrow ((Ri)_{3-0})$

**XRL** <dest-byte>, <src-byte>

**Function:** Logical Exclusive-OR for byte variables  
**Description:** XRL performs the bitwise logical Exclusive-OR operation between the indicated variables, storing the results in the destination. No flags are affected.

The two operands allow six addressing mode combinations. When the destination is the accumulator, the source can use register, direct, register-indirect, or immediate addressing; when the destination is a direct address, the source can be the accumulator or immediate data.

(Note: When this instruction is used to modify an output port, the value used as the original port data will be read from the output data latch, *not* the input pins.)

**Example:** If the accumulator holds 0C3H (11000011B) and register 0 holds 0AAH (10101010B) then the instruction,

XRL A,R0

will leave the accumulator holding the value 69H (01101001B).

When the destination is a directly addressed byte, this instruction can complement combinations of bits in any RAM location or hardware register. The pattern of bits to be complemented is then determined by a mask byte, either a constant contained in the instruction or a variable computed in the accumulator at run-time. The instruction,

XRL P1,#00110001B

will complement bits 5, 4, and 0 of output port 1.

**XRL A,Rn**  
**Bytes:** 1  
**Cycles:** 1

**Encoding:**

0	1	1	0	1	r	r	r
---	---	---	---	---	---	---	---

**Operation:** XRL  
 $(A) \leftarrow (A) \vee (Rn)$

**XRL A,direct**  
**Bytes:** 2  
**Cycles:** 1

**Encoding:**

0	1	1	0	0	1	0	1
---	---	---	---	---	---	---	---

direct address
----------------

**Operation:** XRL  
 $(A) \leftarrow (A) \vee (\text{direct})$



<b>XRL A,@Ri</b>	Bytes: 1	Cycles: 1	Encoding: 0 1 1 0 0 0 1 1 i	Operation: $(A) \leftarrow (A) \vee ((Ri))$	Description: XRL performs the bitwise logical Exclusive-OR operation between the contents of the accumulator and the contents of the register indicated by the register address. The result is stored in the accumulator.	Function: Logical Exclusive-OR for byte variables
<b>XRL A,#data</b>	Bytes: 2	Cycles: 1	Encoding: 0 1 1 0 0 0 1 0 0 immediate data	Operation: $(A) \leftarrow (A) \vee \#data$	Description: XRL performs the bitwise logical Exclusive-OR operation between the contents of the accumulator and the immediate data. The result is stored in the accumulator.	Function: Logical Exclusive-OR for byte variables
<b>XRL direct,A</b>	Bytes: 2	Cycles: 1	Encoding: 0 1 1 0 0 0 0 1 0 direct address	Operation: $(direct) \leftarrow (direct) \vee (A)$	Description: XRL performs the bitwise logical Exclusive-OR operation between the contents of the accumulator and the contents of the memory location indicated by the direct address. The result is stored in the memory location.	Function: Logical Exclusive-OR for byte variables
<b>XRL direct,#data</b>	Bytes: 3	Cycles: 2	Encoding: 0 1 1 0 0 0 0 1 1 direct address immediate data	Operation: $(direct) \leftarrow (direct) \vee \#data$	Description: XRL performs the bitwise logical Exclusive-OR operation between the contents of the memory location indicated by the direct address and the immediate data. The result is stored in the memory location.	Function: Logical Exclusive-OR for byte variables





## 8-BIT CONTROL-ORIENTED MICROCOMPUTERS

8031/8051  
8031AH/8051AH  
8032AH/8052AH  
8751H/8751H-12

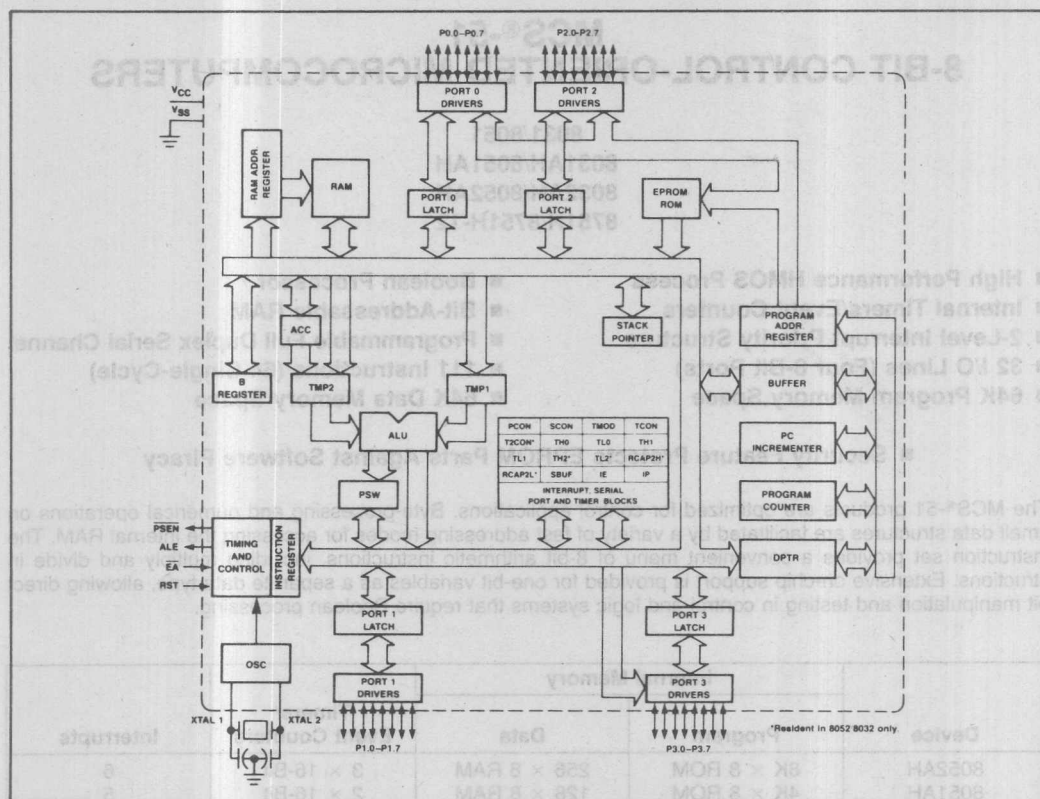
- High Performance HMOS Process
- Internal Timers/Event Counters
- 2-Level Interrupt Priority Structure
- 32 I/O Lines (Four 8-Bit Ports)
- 64K Program Memory Space
- Boolean Processor
- Bit-Addressable RAM
- Programmable Full Duplex Serial Channel
- 111 Instructions (64 Single-Cycle)
- 64K Data Memory Space
- Security Feature Protects EPROM Parts Against Software Piracy

The MCS<sup>®</sup>-51 products are optimized for control applications. Byte-processing and numerical operations on small data structures are facilitated by a variety of fast addressing modes for accessing the internal RAM. The instruction set provides a convenient menu of 8-bit arithmetic instructions, including multiply and divide instructions. Extensive on-chip support is provided for one-bit variables as a separate data type, allowing direct bit manipulation and testing in control and logic systems that require Boolean processing.

Device	Internal Memory		Timers/ Event Counters	Interrupts
	Program	Data		
8052AH	8K × 8 ROM	256 × 8 RAM	3 × 16-Bit	6
8051AH	4K × 8 ROM	128 × 8 RAM	2 × 16-Bit	5
8051	4K × 8 ROM	128 × 8 RAM	2 × 16-Bit	5
8032AH	none	256 × 8 RAM	3 × 16-Bit	6
8031AH	none	128 × 8 RAM	2 × 16-Bit	5
8031	none	128 × 8 RAM	2 × 16-Bit	5
8751H	4K × 8 EPROM	128 × 8 RAM	2 × 16-Bit	5
8751H-12	4K × 8 EPROM	128 × 8 RAM	2 × 16-Bit	5

The 8751H is an EPROM version of the 8051AH; that is, the on-chip Program Memory can be electrically programmed, and can be erased by exposure to ultraviolet light. It is fully compatible with its predecessor, the 8751-8, but incorporates two new features: a Program Memory Security bit that can be used to protect the EPROM against unauthorized read-out, and a programmable baud rate modification bit (SMOD). SMOD is not present in the 8751H-12.





**Figure 1. MCS<sup>®</sup>-51 Block Diagram**

## PIN DESCRIPTIONS

## VCC

Supply voltage.

## VSS

Circuit ground.

## Port 0

Port 0 is an 8-bit open drain bidirectional I/O port. As an output port each pin can sink 8 LS TTL inputs. Port 0 pins that have 1s written to them float, and in that state can be used as high-impedance inputs.

Port 0 is also the multiplexed low-order address and data bus during accesses to external Program and Data Memory. In this application it uses strong internal pullups when emitting 1s, and can source and sink 8 LS TTL inputs.

Port 0 also receives the code bytes during programming of the EPROM parts, and outputs the code bytes during program verification of the ROM and EPROM parts. External pullups are required during program verification.

## Port 1

Port 1 is an 8-bit bidirectional I/O port with internal pullups. The Port 1 output buffers can sink/source 4 LS TTL inputs. Port 1 pins that have 1s written to them are pulled high by the internal pullups, and in that state can be used as inputs. As inputs, Port 1 pins that are externally being pulled low will source current (IIL, on the data sheet) because of the internal pullups.

Port 1 also receives the low-order address bytes during programming of the EPROM parts and during program verification of the ROM and EPROM parts.

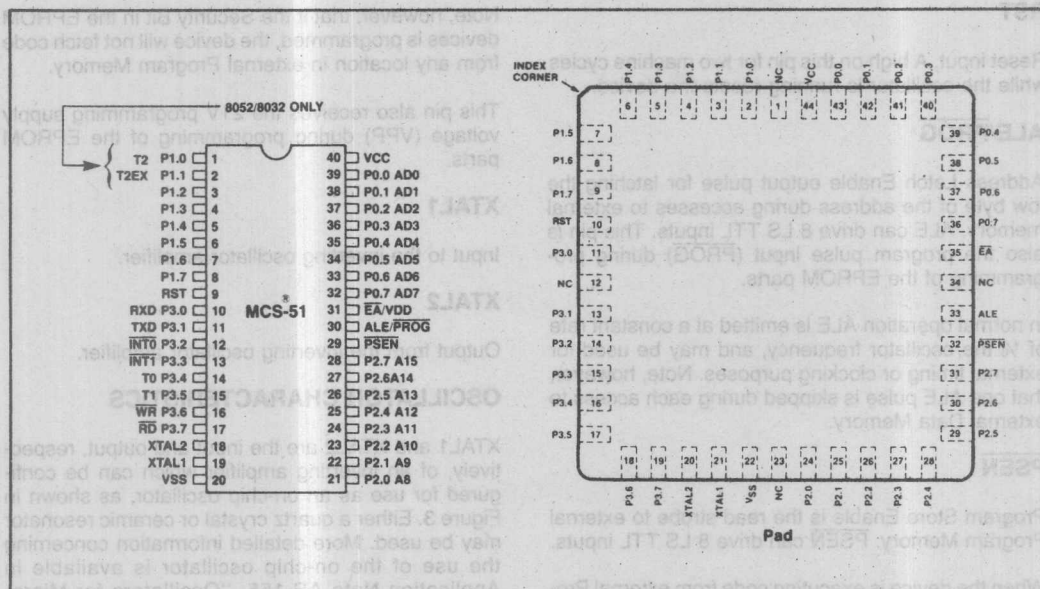


Figure 2. MCS<sup>®</sup>-51 Pin Connections

In the 8032AH and 8052AH, Port 1 pins P1.0 and P1.1 also serve the T2 and T2EX functions, respectively.

## Port 2

Port 2 is an 8-bit bidirectional I/O port with internal pullups. The Port 2 output buffers can sink/source 4 LS TTL inputs. Port 2 pins that have 1s written to them are pulled high by the internal pullups, and in that state can be used as inputs. As inputs, Port 2 pins that are externally being pulled low will source current (IIL, on the data sheet) because of the internal pullups.

Port 2 emits the high-order address byte during fetches from external Program Memory and during accesses to external Data Memory that use 16-bit addresses (MOVX @DPTR). In this application it uses strong internal pullups when emitting 1s. During accesses to external Data Memory that use 8-bit addresses (MOVX @Ri), Port 2 emits the contents of the P2 Special Function Register.

Port 2 also receives the high-order address bits during programming of the EPROM parts and during program verification of the ROM and EPROM parts.

## Port 3

Port 3 is an 8-bit bidirectional I/O port with internal pullups. The Port 3 output buffers can sink/source 4 LS TTL inputs. Port 3 pins that have 1s written to them are pulled high by the internal pullups, and in that state can be used as inputs. As inputs, Port 3 pins that are externally being pulled low will source current (IIL, on the data sheet) because of the pullups.

Port 3 also serves the functions of various special features of the MCS-51 Family, as listed below:

Port Pin	Alternative Function
P3.0	RXD (serial input port)
P3.1	TXD (serial output port)
P3.2	INT0 (external interrupt 0)
P3.3	INT1 (external interrupt 1)
P3.4	T0 (Timer 0 external input)
P3.5	T1 (Timer 1 external input)
P3.6	WR (external data memory write strobe)
P3.7	RD (external data memory read strobe)

## RST

Reset input. A high on this pin for two machine cycles while the oscillator is running resets the device.

## ALE/PROG

Address Latch Enable output pulse for latching the low byte of the address during accesses to external memory. ALE can drive 8 LS TTL inputs. This pin is also the program pulse input (PROG) during programming of the EPROM parts.

In normal operation ALE is emitted at a constant rate of  $\frac{1}{6}$  the oscillator frequency, and may be used for external timing or clocking purposes. Note, however, that one ALE pulse is skipped during each access to external Data Memory.

## PSEN

Program Store Enable is the read strobe to external Program Memory. PSEN can drive 8 LS TTL inputs.

When the device is executing code from external Program Memory, PSEN is activated twice each machine cycle, except that two PSEN activations are skipped during each access to external Data Memory.

## EA/VPP

External Access enable  $\overline{EA}$  must be externally held low in order to enable any MCS-51 device to fetch code from external Program Memory locations 0 to 0FFFF (0 to 1FFFF, in the 8032AH and 8052AH).

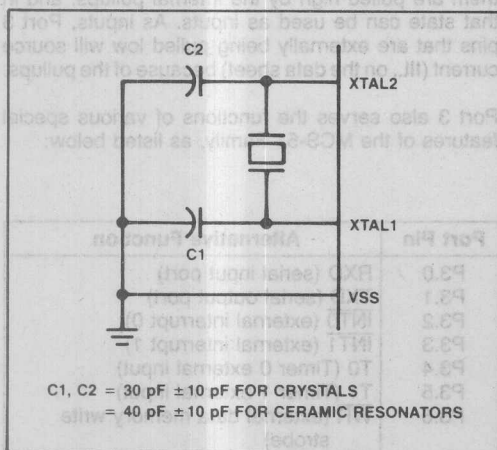


Figure 3. Oscillator Connections

Note, however, that if the Security Bit in the EPROM devices is programmed, the device will not fetch code from any location in external Program Memory.

This pin also receives the 21V programming supply voltage (VPP) during programming of the EPROM parts.

## XTAL1

Input to the inverting oscillator amplifier.

## XTAL2

Output from the inverting oscillator amplifier.

## OSCILLATOR CHARACTERISTICS

XTAL1 and XTAL2 are the input and output, respectively, of an inverting amplifier which can be configured for use as an on-chip oscillator, as shown in Figure 3. Either a quartz crystal or ceramic resonator may be used. More detailed information concerning the use of the on-chip oscillator is available in Application Note AP-155, "Oscillators for Micro-controllers."

To drive the device from an external clock source, XTAL1 should be grounded, while XTAL2 is driven, as shown in Figure 4. There are no requirements on the duty cycle of the external clock signal, since the input to the internal clocking circuitry is through a divide-by-two flip-flop, but minimum and maximum high and low times specified on the Data Sheet must be observed.

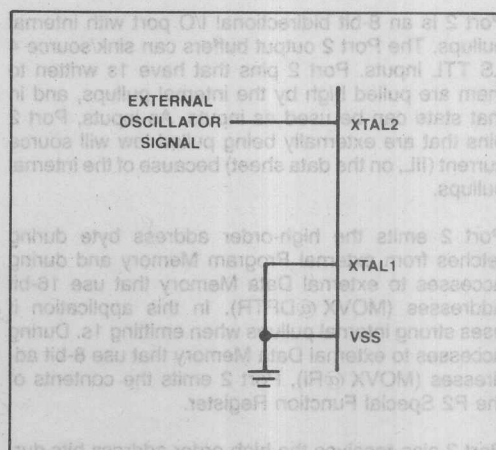


Figure 4. External Drive Configuration

## ABSOLUTE MAXIMUM RATINGS\*

Ambient Temperature Under Bias . . . 0 °C to 70 °C  
 Storage Temperature . . . . . -65 °C to +150 °C  
 Voltage on  $\overline{\text{EA}}$ /VPP Pin to VSS . -0.5V to +21.5V  
 Voltage on Any Other Pin to VSS . -0.5V to +7V  
 Power Dissipation . . . . . 1.5W

\*NOTICE: Stresses above those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.

## D.C. CHARACTERISTICS: ( $T_A = 0^\circ\text{C}$ to $70^\circ\text{C}$ ; $V_{CC} = 5V \pm 10\%$ ; $V_{SS} = 0V$ )

Symbol	Parameter	Min	Max	Unit	Test Conditions
VIL	Input Low Voltage (Except $\overline{\text{EA}}$ Pin of 8751H, 8751H-12)	-0.5	0.8	V	
VIL1	Input Low Voltage to $\overline{\text{EA}}$ Pin of 8751H, 8751H-12	0	0.7	V	
VIH	Input High Voltage (Except XTAL2, RST)	2.0	$V_{CC} + 0.5$	V	
VIH1	Input High Voltage to XTAL2, RST	2.5	$V_{CC} + 0.5$	V	XTAL1 = VSS
VOL	Output Low Voltage (Ports 1, 2, 3)*		0.45	V	IOL = 1.6 mA
VOL1	Output Low Voltage (Port 0, ALE, PSEN)*				
			0.60	V	IOL = 3.2 mA
	8751H, 8751H-12		0.45	V	IOL = 2.4 mA
	All Others		0.45	V	IOL = 3.2 mA
VOH	Output High Voltage (Ports 1, 2, 3)	2.4		V	IOH = -80 $\mu\text{A}$
VOH1	Output High Voltage (Port 0 in External Bus Mode, ALE, PSEN)	2.4		V	IOH = -400 $\mu\text{A}$
IIL	Logical 0 Input Current (Ports 1, 2, 3)				
	8032AH, 8052AH All Others		-800 -500	$\mu\text{A}$ $\mu\text{A}$	Vin = 0.45 V Vin = 0.45 V
IIL1	Logical 0 Input Current to $\overline{\text{EA}}$ Pin of 8751H, 8751H-12 Only		-15	mA	
IIL2	Logical 0 Input Current (XTAL2)		-3.2	mA	Vin = 0.45 V
ILI	Input Leakage Current (Port 0)				
	8751H, 8751H-12 All Others		$\pm 100$ $\pm 10$	$\mu\text{A}$ $\mu\text{A}$	$0.45 < V_{in} < V_{CC}$ $0.45 < V_{in} < V_{CC}$
IIH	Logical 1 Input Current to $\overline{\text{EA}}$ Pin of 8751H, 8751H-12		500	$\mu\text{A}$	
IIH1	Input Current to RST to Activate Reset		500	$\mu\text{A}$	Vin < (VCC - 1.5V)
ICC	Power Supply Current: 8031/8051		160	mA	All Outputs Disconnected; $\overline{\text{EA}} = V_{CC}$
	8031AH/8051AH		125	mA	
	8032AH/8052AH		175	mA	
	8751H/8751H-12		250	mA	
CIO	Pin Capacitance		10	pF	test freq = 1MHz

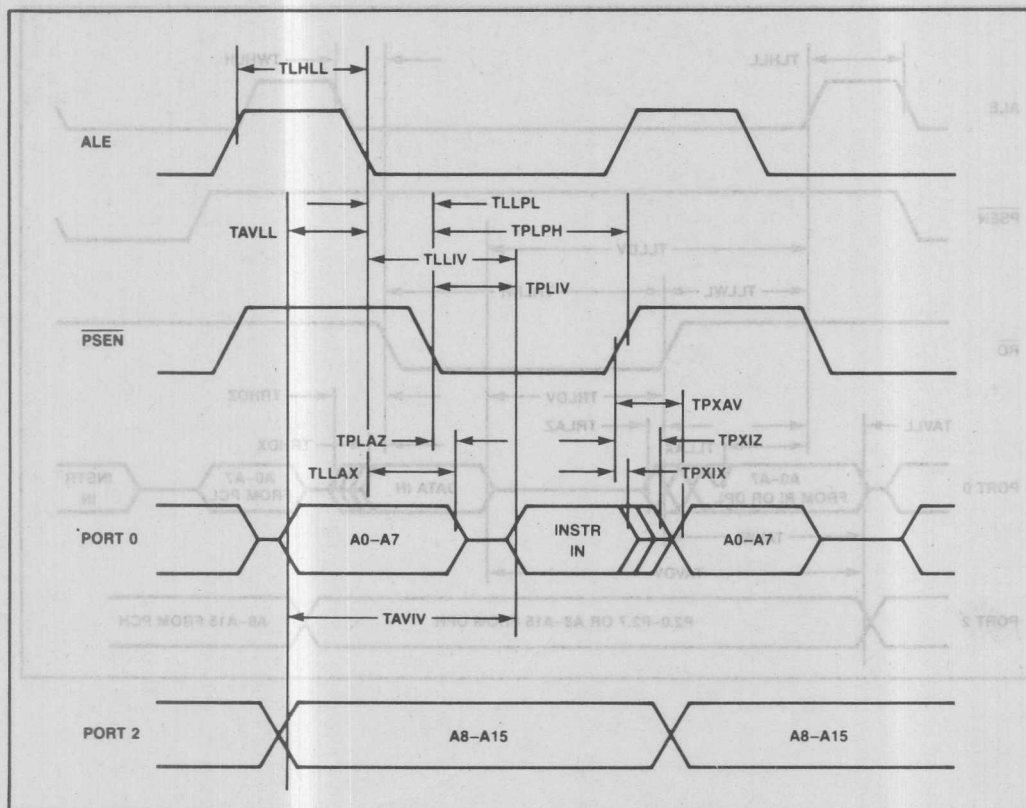
\*Note: Capacitive loading on Ports 0 and 2 may cause spurious noise pulses to be superimposed on the VOLs of ALE and Ports 1 and 3. The noise is due to external bus capacitance discharging into the Port 0 and Port 2 pins when these pins make 1-to-0 transitions during bus operations. In the worst cases (capacitive loading > 100 pF), the noise pulse on the ALE line may exceed 0.8V. In such cases it may be desirable to qualify ALE with a Schmitt Trigger, or use an address latch with a Schmitt Trigger STROBE input.



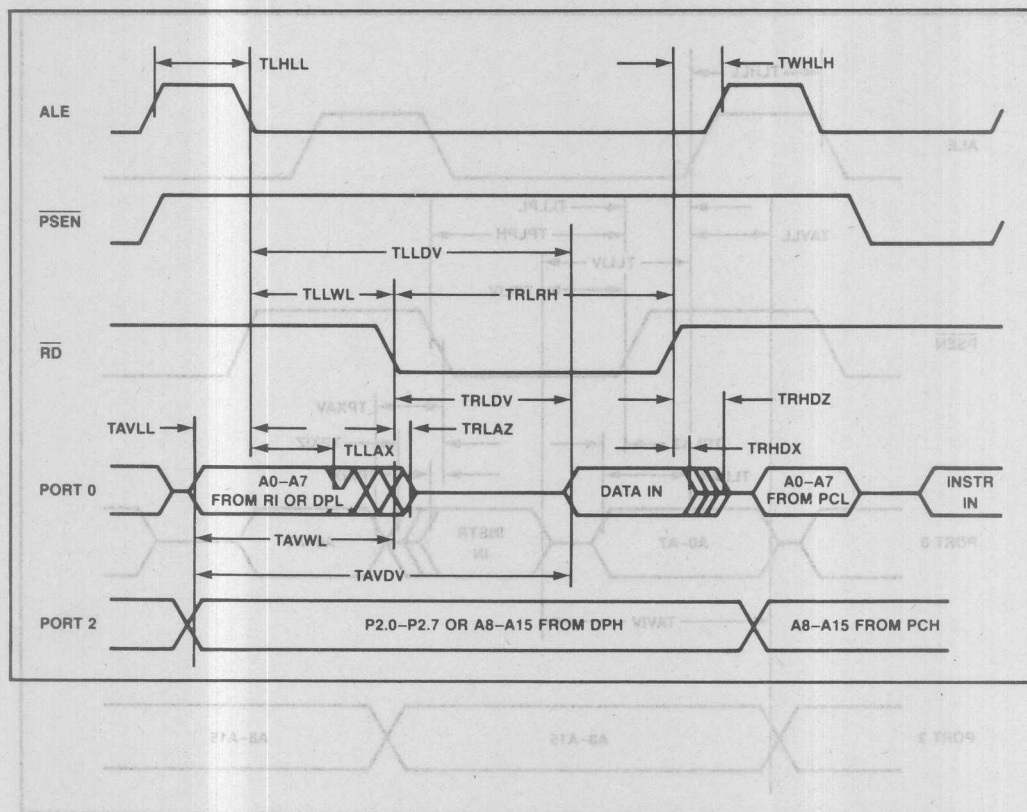
**A.C. CHARACTERISTICS:** ( $T_A = 0^\circ\text{C}$  to  $+70^\circ\text{C}$ ,  $V_{CC} = 5V \pm 10\%$ ,  $V_{SS} = 0V$ ,  
Load Capacitance for Port 0, ALE, and PSEN = 100 pF,  
Load Capacitance for All Other Outputs = 80 pF)

Symbol	Parameter	12MHz Osc		Variable Oscillator		Units
		Min	Max	Min	Max	
1/TCLCL	Oscillator Frequency			3.5	12	MHz
TLHLL	ALE Pulse Width	127		2TCLCL-40		ns
TAVLL	Address Valid to ALE Low	43		TCLCL-40		ns
TLLAX	Address Hold After ALE Low	48		TCLCL-35		ns
TLLIV	ALE Low to Valid Instr In 8751H, 8751H-12 All Others		183 233		4TCLCL-150 4TCLCL-100	ns
TLLPL	ALE Low to PSEN Low	58		TCLCL-25		ns
TPLPH	PSEN Pulse Width 8751H, 8751H-12 All Others	190 215		3TCLCL-60 3TCLCL-35		ns ns
TPLIV	PSEN Low to Valid Instr In 8751H, 8751H-12 All Others		100 125		3TCLCL-150 3TCLCL-125	ns ns
TPXIX	Input Instr Hold After PSEN	0		0		ns
TPXIZ	Input Instr Float After PSEN		63		TCLCL-20	ns
TPXAV	PSEN to Address Valid	75		TCLCL-8		ns
TAVIV	Address to Valid Instr In 8751H, 8751H-12 All Others		267 302		5TCLCL-150 5TCLCL-115	ns ns
TPLAZ	PSEN Low to Address Float		TBD		TBD	ns
TRLRH	RD Pulse Width	400		6TCLCL-100		ns
TWLWH	WR Pulse Width	400		6TCLCL-100		ns
TRLDV	RD Low to Valid Data In		252		5TCLCL-165	ns
TRHDX	Data Hold After RD	0		0		ns
TRHDZ	Data Float After RD		97		2TCLCL-70	ns
TLLDV	ALE Low to Valid Data In		517		8TCLCL-150	ns
TAVDV	Address to Valid Data In		585		9TCLCL-165	ns
TLLWL	ALE Low to RD or WR Low	200	300	3TCLCL-50	3TCLCL + 50	ns
TAVWL	Address to RD or WR Low	203		4TCLCL-130		ns
TQVWX	Data Valid to WR Transition 8751H, 8751H-12 All Others	13 23		TCLCL-70 TCLCL-60		ns ns
TQVWH	Data Valid to WR High	433		7TCLCL-150		ns
TWHQX	Data Held After WR	33		TCLCL-50		ns
TRLAZ	RD Low to Address Float		TBD		TBD	ns
TWHLH	RD or WR High to ALE High 8751H, 8751H-12 All Others	33 43	133 123	TCLCL-50 TCLCL-40	TCLCL + 50 TCLCL + 40	ns ns

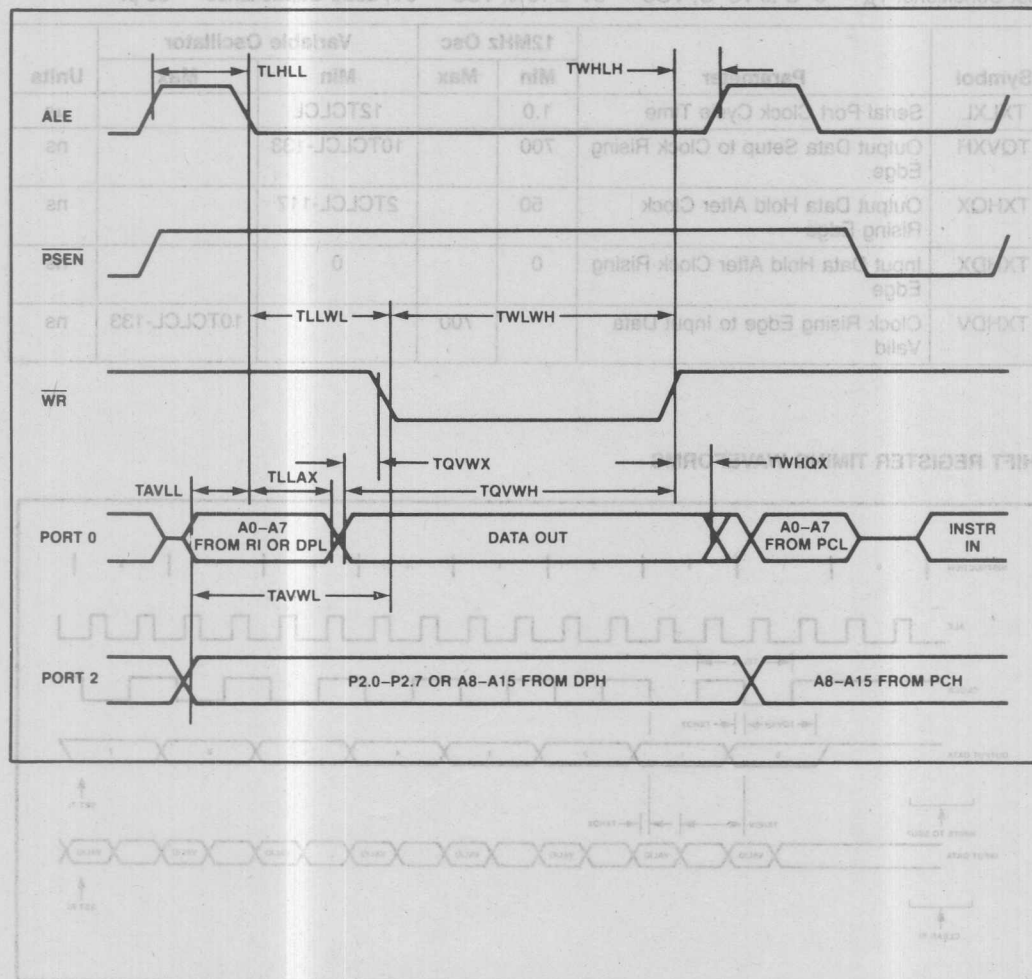
EXTERNAL PROGRAM MEMORY READ CYCLE



EXTERNAL DATA MEMORY READ CYCLE



EXTERNAL DATA MEMORY WRITE CYCLE



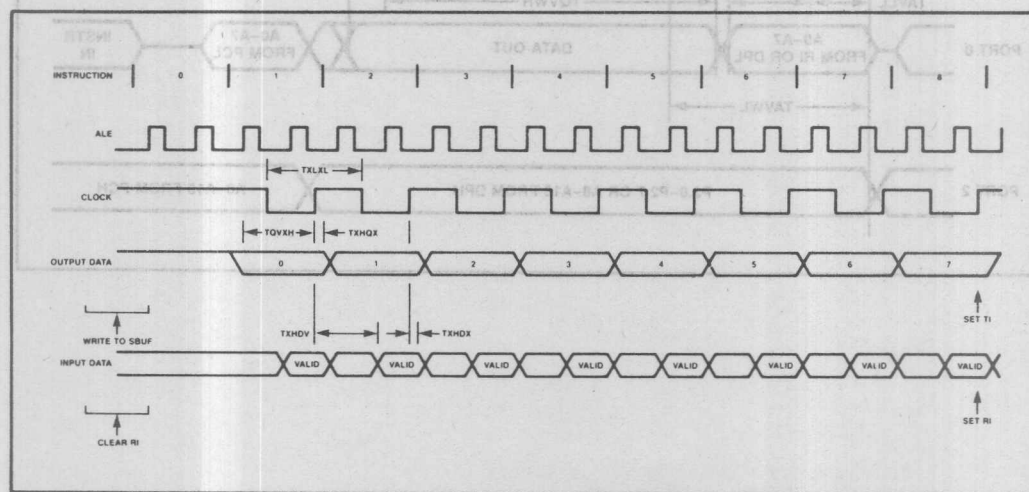


# SERIAL PORT TIMING — SHIFT REGISTER MODE

Test Conditions:  $T_A = 0^\circ\text{C}$  to  $70^\circ\text{C}$ ;  $V_{CC} = 5\text{V} \pm 10\%$ ;  $V_{SS} = 0\text{V}$ ; Load Capacitance =  $80\text{ pF}$

Symbol	Parameter	12MHz Osc		Variable Oscillator		Units
		Min	Max	Min	Max	
TXLXL	Serial Port Clock Cycle Time	1.0		12TCLCL		$\mu\text{s}$
TQVXH	Output Data Setup to Clock Rising Edge	700		10TCLCL-133		ns
TXHQX	Output Data Hold After Clock Rising Edge	50		2TCLCL-117		ns
TXHDX	Input Data Hold After Clock Rising Edge	0		0		ns
TXHDV	Clock Rising Edge to Input Data Valid		700		10TCLCL-133	ns

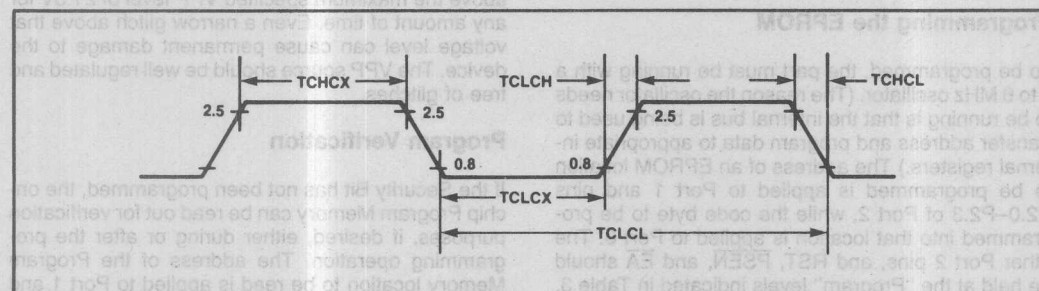
## SHIFT REGISTER TIMING WAVEFORMS



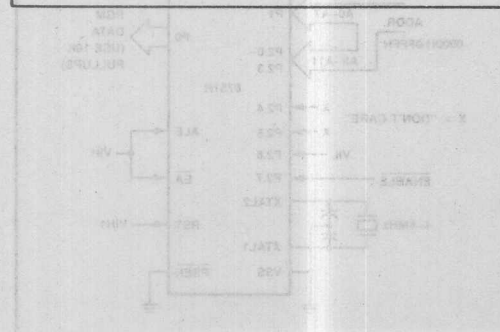
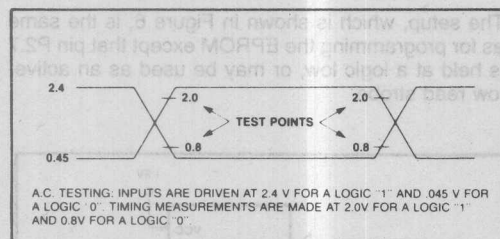
# EXTERNAL CLOCK DRIVE

Symbol	Parameter	Min	Max	Units
1/TCLCL	Oscillator Frequency	3.5	12	MHz
TCHCX	High Time	20		ns
TCLCX	Low Time	20		ns
TCLCH	Rise Time	0	20	ns
TCHCL	Fall Time	0	20	ns

## EXTERNAL CLOCK DRIVE WAVEFORMS



## A.C. TESTING INPUT, OUTPUT WAVEFORM



# EPROM CHARACTERISTICS:

Table 3. EPROM Programming Modes

Mode	RST	PSEN	ALE	EA	P2.7	P2.6	P2.5	P2.4
Program	1	0	0*	VPP	1	0	X	X
Inhibit	1	0	1	X	1	0	X	X
Verify	1	0	1	1	0	0	X	X
Security Set	1	0	0*	VPP	1	1	X	X

Note: "1" = logic high for that pin  
"0" = logic low for that pin  
"X" = "don't care"  
"VPP" = +21V  $\pm$  0.5V

\*ALE is pulsed low for 50 msec.

## Programming the EPROM

To be programmed, the part must be running with a 4 to 6 MHz oscillator. (The reason the oscillator needs to be running is that the internal bus is being used to transfer address and program data to appropriate internal registers.) The address of an EPROM location to be programmed is applied to Port 1 and pins P2.0–P2.3 of Port 2, while the code byte to be programmed into that location is applied to Port 0. The other Port 2 pins, and RST, PSEN, and EA should be held at the "Program" levels indicated in Table 3. ALE is **pulsed** low for 50 msec to program the code byte into the addressed EPROM location. The setup is shown in Figure 5.

Normally EA is held at a logic high until just before ALE is to be pulsed. Then EA is raised to +21V, ALE is pulsed, and then EA is returned to a logic high. Waveforms and detailed timing specifications are shown in later sections of this data sheet.

Note that the EA/VPP pin must not be allowed to go above the maximum specified VPP level of 21.5V for any amount of time. Even a narrow glitch above that voltage level can cause permanent damage to the device. The VPP source should be well regulated and free of glitches.

## Program Verification

If the Security Bit has not been programmed, the on-chip Program Memory can be read out for verification purposes, if desired, either during or after the programming operation. The address of the Program Memory location to be read is applied to Port 1 and pins P2.0–P2.3. The other pins should be held at the "Verify" levels indicated in Table 3. The contents of the addressed location will come out on Port 0. External pullups are required on Port 0 for this operation.

The setup, which is shown in Figure 6, is the same as for programming the EPROM except that pin P2.7 is held at a logic low, or may be used as an active-low read strobe.

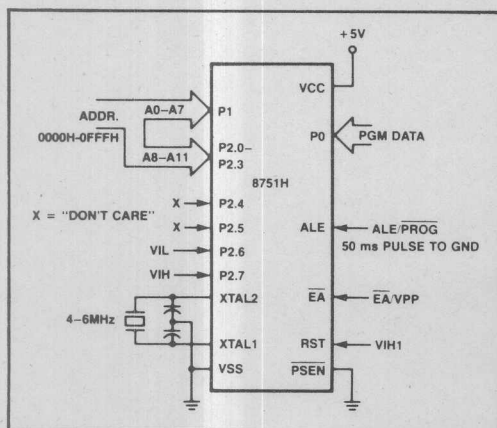


Figure 5. Programming Configuration

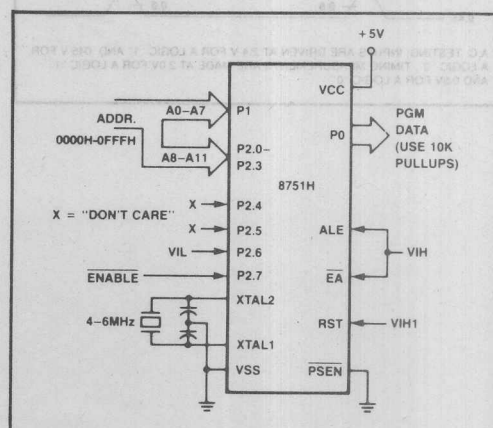
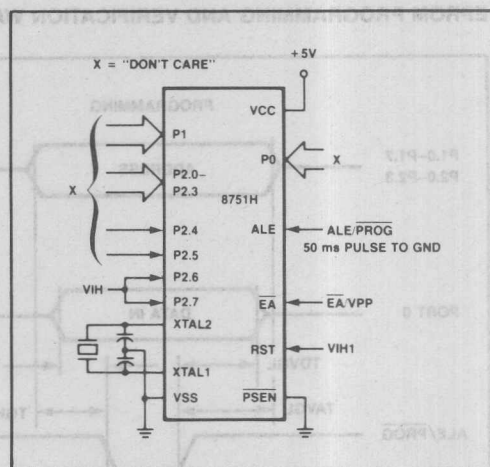


Figure 6. Program Verification

## EPROM Security

The security feature consists of a "locking" bit which when programmed denies electrical access by any external means to the on-chip Program Memory. The bit is programmed as shown in Figure 7. The setup and procedure are the same as for normal EPROM programming, except that P2.6 is held at a logic high. Port 0, Port 1, and pins P2.0–P2.3 may be in any state. The other pins should be held at the "Security" levels indicated in Table 3.

Once the Security Bit has been programmed, it can be cleared only by full erasure of the Program Memory. While it is programmed, the internal Program Memory can not be read out, the device can not be further programmed, and it **can not execute out of external program memory**. Erasing the EPROM, thus clearing the Security Bit, restores the device's full functionality. It can then be reprogrammed.



### Figure 7. Programming the Security Bit

### Erasure Characteristics

Erasure of the EPROM begins to occur when the chip is exposed to light with wavelengths shorter than approximately 4,000 Angstroms. Since sunlight and fluorescent lighting have wavelengths in this range, exposure to these light sources over an extended time (about 1 week in sunlight, or 3 years in room-level fluorescent lighting) could cause inadvertent erasure. If an application subjects the device to this type of exposure, it is suggested that an opaque label be placed over the window.

The recommended erasure procedure is exposure to ultraviolet light (at 2537 Angstroms) to an integrated dose of at least 15 W-sec/cm<sup>2</sup>. Exposing the EPROM to an ultraviolet lamp of 12,000  $\mu$ W/cm<sup>2</sup> rating for 20 to 30 minutes, at a distance of about 1 inch, should be sufficient.

Erasure leaves the array in an all 1s state.

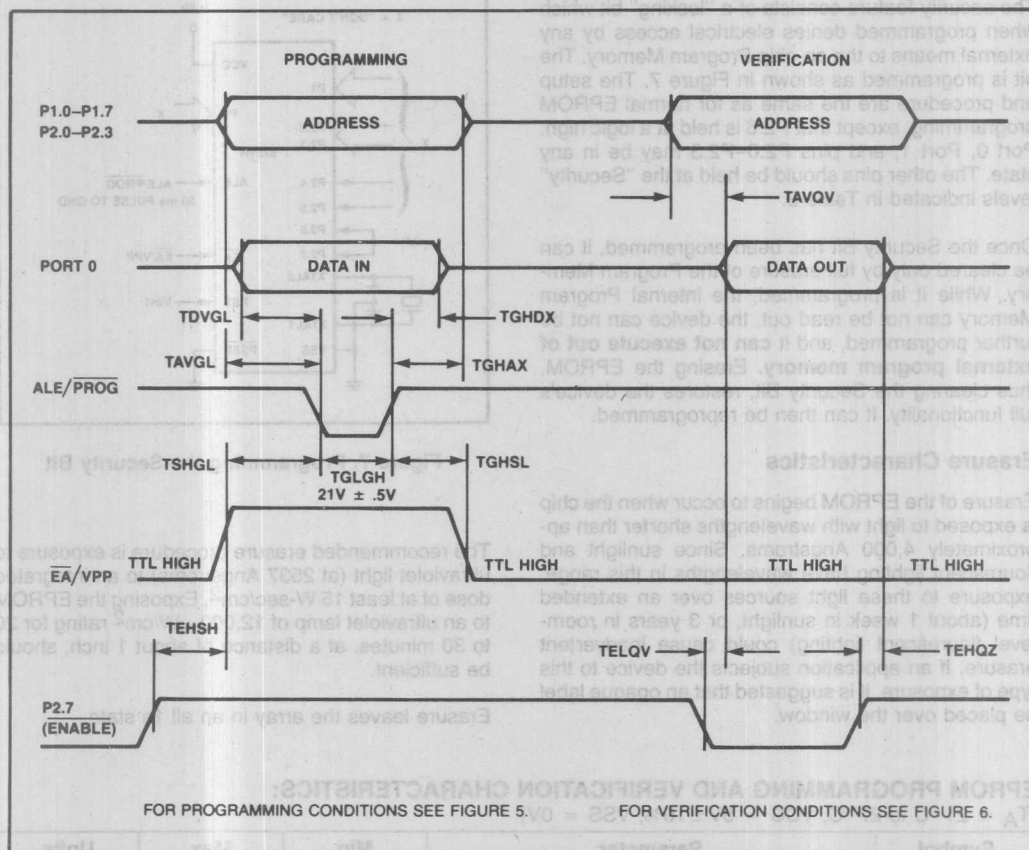
#### EPROM PROGRAMMING AND VERIFICATION CHARACTERISTICS:

(T<sub>A</sub> = 21 °C to 27 °C, VCC = 5V ± 10%, VSS = 0V)

Symbol	Parameter	Min	Max	Units
VPP	Programming Supply Voltage	20.5	21.5	V
IPP	Programming Supply Current		30	mA
1/TCLCL	Oscillator Frequency	4	6	MHz
TAVGL	Address Setup to $\overline{\text{PROG}}$ Low	48TCLCL		
TGHAX	Address Hold After $\overline{\text{PROG}}$	48TCLCL		
TDVGL	Data Setup to $\overline{\text{PROG}}$ Low	48TCLCL		
TGHDX	Data Hold After $\overline{\text{PROG}}$	48TCLCL		
TEHSH	P2.7 ( $\overline{\text{ENABLE}}$ ) High to VPP	48TCLCL		
TSHGL	VPP Setup to $\overline{\text{PROG}}$ Low	10		$\mu\text{sec}$
TGHSL	VPP Hold After $\overline{\text{PROG}}$	10		$\mu\text{sec}$
TGLGH	$\overline{\text{PROG}}$ Width	45	55	msec
TAVQV	Address to Data Valid		48TCLCL	
TELQV	$\overline{\text{ENABLE}}$ Low to Data Valid		48TCLCL	
TEHQZ	Data Float After $\overline{\text{ENABLE}}$	0	48TCLCL	



# EPROM PROGRAMMING AND VERIFICATION WAVEFORMS



Symbol	Parameter	Min	Max	Unit
VPP	Programming Supply Voltage	20.5	27.5	V
IPP	Programming Supply Current	30		mA
fTCLC	Oscillator Frequency	4		MHz
TAVL	Address Setup to PROG Low	48TCLC		
TGHAX	Address Hold After PROG	48TCLC		
TDVGL	Data Setup to PROG Low	48TCLC		
TGHDX	Data Hold After PROG	48TCLC		
TEHSH	P2.7 (ENABLE) High to VPP	48TCLC		
TSHGL	VPP Setup to PROG Low	10		μsec
TGHSL	VPP Hold After PROG	10		μsec
TGLGH	PROG Width	48	55	μsec
TAVQV	Address to Data Valid	48TCLC		
TELQV	ENABLE Low to Data Valid	48TCLC		
TEHQZ	Data First After ENABLE	0		

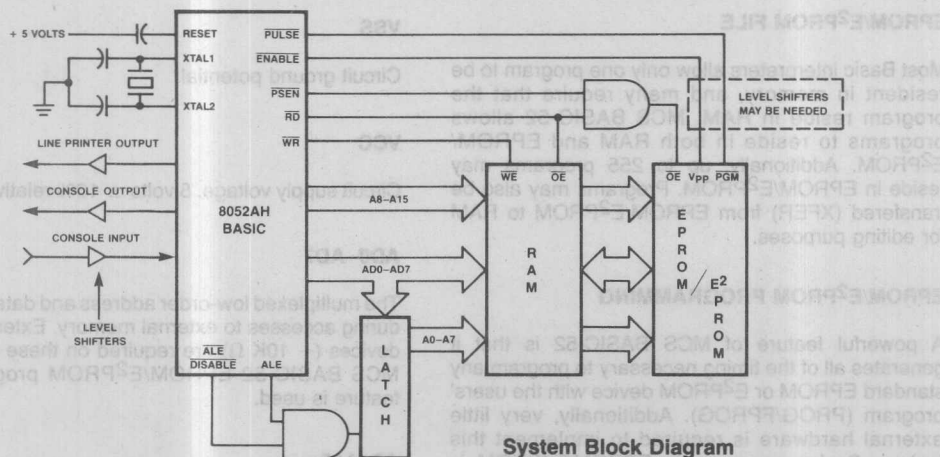
- Full BASIC Interpreter in ROM on a Single Chip
- BCD Floating Point Math
- Generates All Timing Necessary to Program EPROMS and E<sup>2</sup>PROMS
- Fast Tokenized Interpreter
- "Stand Alone" Software Development
- All Arithmetic and Utility Routines Can Be Called From Assembly Language
- Interrupts Can Be Handled By BASIC or Assembly Language
- Built-In Accurate REAL TIME CLOCK
- Multiple User Programs
- Programs May Reside in RAM, EPROM or E<sup>2</sup>PROM
- Built in Radix Conversion — Hex to Decimal and Decimal to Hex

8052AH-BASIC is an 8052AH microcontroller with a complete full-featured BASIC interpreter, MCS® BASIC-52, resident in the 8K of available ROM. This Software-On-Silicon product is specifically designed to address the needs of process control, measurement, and instrumentation applications. MCS BASIC-52 allows 8052AH users to write programs in the popular BASIC language, which is much simpler to write and easier to understand than assembly language.

In addition to the standard BASIC commands and functions, such as floating point arithmetic and transcendental operations, MCS BASIC-52 contains many unique features that allow the user to perform tasks that usually require assembly language. Bit-wise logical operators, such as AND, OR, and EXCLUSIVE-OR are supported as well as hexadecimal arithmetic.

A minimum amount of hardware is required to support MCS BASIC-52. Small systems can be constructed with only a latch, 1K bytes of external memory, and the appropriate serial port drivers. With the addition of a transistor, a gate, and a couple of passive components, MCS BASIC-52 can program EPROM or E<sup>2</sup>PROM devices with the users application program. Both the standard and the intelligent Programming™ algorithms are supported.

MCS BASIC-52 is an interpreted language. This allows the user to develop a program interactively without the cumbersome and repetitive process of editing, assembling, loading, and running which is required by assemblers and compilers. MCS BASIC-52 was designed to permit the programmer to develop resident high level language software using the high performance 8052AH device.



## COMMAND SET

MCS BASIC-52 contains all standard BASIC commands, statements, and operators. Figure 1 list the software feature set of MCS BASIC-52.

## DATA FORMAT

The range of numbers that can be represented in MCS BASIC-52 is:

$$\pm 1E-127 \text{ to } \pm .99999999E + 127$$

## CONTROL ORIENTED FEATURES

MCS BASIC-52 contains many unique features to perform task that usually require assembly language programming. The XBY and DBY operators can read and/or write external and internal memory respectively. The CBY operator is used to read program memory. Additionally, virtually all of the special function registers on the 8052AH can be accessed with MCS BASIC-52. This allows the user to set the timer or interrupt modes within the constructs of a BASIC program. An accurate interrupt driven REAL TIME CLOCK that has a 5 millisecond resolution is also implemented in MCS BASIC-52. This clock can be enabled, disabled, and used to generate interrupts. Finally, a CALL statement that allows the programmer to CALL assembly language routines is available in MCS BASIC-52. Parameters can be passed in a number of different ways.

## EPROM/E<sup>2</sup>PROM FILE

Most Basic interpreters allow only one program to be resident in memory, and many require that the program reside in RAM. MCS BASIC-52 allows programs to reside in both RAM and EPROM/E<sup>2</sup>PROM. Additionally, up to 255 programs may reside in EPROM/E<sup>2</sup>PROM. Programs may also be transferred (XFER) from EPROM/E<sup>2</sup>PROM to RAM for editing purposes.

## EPROM/E<sup>2</sup>PROM PROGRAMMING

A powerful feature of MCS BASIC-52 is that it generates all of the timing necessary to program any standard EPROM or E<sup>2</sup>PROM device with the users' program (PROG/FPROG). Additionally, very little external hardware is required to implement this feature. Saving programs in EPROM/E<sup>2</sup>PROM is much more attractive and reliable than other alternatives, such as cassette tape, especially in control and/or other noisy environments.

After the user programs an EPROM or E<sup>2</sup>PROM with the desired BASIC program. The PROG2 or FPROG2 commands may be used to enable the unique AUTOSTART feature of MCS BASIC-52. When AUTOSTART is enabled, MCS BASIC-52 will execute the user program after RESET or a power-up condition. This permits the user to RUN a program without connecting the MCS BASIC-52 device to a console — a powerful feature for control environments.

## USER ACCESSABLE FUNCTION LIBRARY

Another unique feature of MCS BASIC-52 is that it contains a complete library of functions that can be accessed with assembly language. All floating point, radix conversion, and I/O routines contained in MCS BASIC-52 can be accessed with assembly language CALL instructions. These complex arithmetic routines can be used by the programmer in applications requiring the speed of assembly language, but also the complex arithmetics offered by BASIC.

## 8052AH-BASIC PIN DESCRIPTION (FIGURE 2)

8052AH-BASIC is an 8052AH device, however, MCS BASIC-52 assumes a particular hardware configuration. The following pin description outlines the pin functions defined by MCS BASIC-52.

### VSS

Circuit ground potential.

### VCC

Circuit supply voltage. 5 volts  $\pm$  10% relative to VSS.

### AD0-AD7

The multiplexed low-order address and data bus used during accesses to external memory. External pullup devices ( $\sim 10K \Omega$ ) are required on these pins if the MCS BASIC-52 EPROM/E<sup>2</sup>PROM programming feature is used.

### A8-A15

The high order address bus used during accesses to external memory.

Commands	Statements	Operators
RUN	BAUD	ADD (+)
LIST	CALL	DIVIDE (/)
LIST#	CLEAR	EXPONENTIATION (**)
NEW	CLEAR	MULTIPLY (*)
NULL	CLEAR	SUBTRACT (-)
RAM	CLOCK0	LOGICAL AND (.AND.)
ROM	CLOCK1	LOGICAL OR (.OR.)
XFER	DATA	LOGICAL X-OR (.XOR.)
PROG	READ	LOGICAL NOT
PROG1	RESTORE	ABS ( )
PROG2	DIM	INT ( )
FPROG	DO-WHILE	SGN ( )
FPROG1	DO-UNTIL	SQR ( )
FPROG2	END	RND
	FOR-TO-STEP	LOG ( )
	NEXT	EXP ( )
	GOSUB	SIN ( )
	RETURN	COS ( )
	GOTO	TAN ( )
	ON-GOTO	ATN ( )
	ON-GOSUB	=, >, >=, <, <=, <>
	IF-THEN-ELSE	ASC ( )
	INPUT	CHR ( )
	LET	CBY ( )
	ONERR	DBY ( )
	ONEXT1	XBY ( )
	ONTIME	GET
	PRINT	IE
	PRINT#	IP
	PH0.	PORT1
	PH0.#	PCON
	PH1.	RCAP2
	PH1.#	T2CON
	PUSH	TCON
	POP	TMOD
	PWM	TIME
	REM	TIMER0
	RET1	TIMER1
	STOP	TIMER2
	STRING	TIME
	UI0	XTAL
	UI1	MTOP
	UO0	LEN
	UO1	FREE
		PI

Figure 1. MCS® BASIC-52 Software Feature Set

## PORT 1

A general purpose quasi-bidirectional 8-bit input/output port. The individual pins on PORT 1 all have alternate functions which may or may not be implemented by the user. The alternate functions are as follows:

### PORT 1.0 (T2)

Can be used as the trigger input to TIMER/COUNTER 2. A one (1) must be written to this port pin output latch in order for this function to operate. Details of

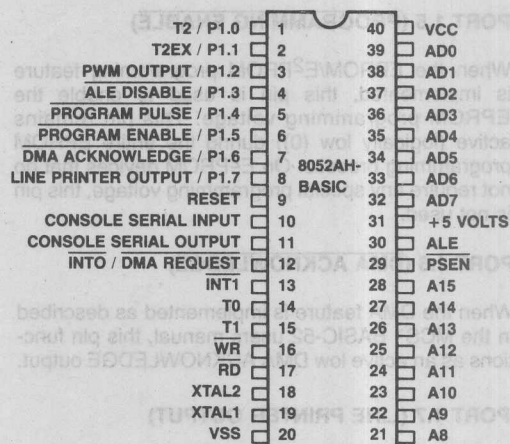


Figure 2. Configuration

the T2 trigger function are covered in the Microcontrollers Handbook. Order Number 210918-002.

### PORT 1.1 (T2EX)

Can be used as the external input to TIMER/COUNTER 2. A one (1) must be written to this port pin output latch in order for this function to operate. Details of the T2 trigger function are covered in the Microcontroller Users Manual.

### PORT 1.2 (PWM OUTPUT)

This pin is used as the PWM output port when the PWM statement is executed. PWM stands for Pulse Width Modulation and is used to generate pulses of varying duty cycle and frequency.

### PORT 1.3 (ALE DISABLE)

This pin is used to disable the ALE signal to the external address latch when the EPROM/E<sup>2</sup>PROM programming feature is used. In a system, this pin is logically anded with ALE.

### PORT 1.4 (PROGRAMMING PULSE)

When the EPROM/E<sup>2</sup>PROM programming feature is used, this pin provides the proper programming pulse width to program EPROM and INTELlgent EPROM® devices. MCS BASIC-52 actually calculates the proper programming pulse width from the system crystal value (XTAL) to assure the proper timing of this pulse. When used to program E<sup>2</sup>PROM devices, the length of this pulse is not critical. This pin is active in the logical zero (0) state.



**PORT 1.5 (PROGRAMMING ENABLE)**

When the EPROM/E<sup>2</sup>PROM programming feature is implemented, this pin is used to enable the EPROM programming voltage. This pin remains active (logically low (0)) during the entire EPROM programming process. On E<sup>2</sup>PROM devices that do not require any special programming voltage, this pin is not used.

**PORT 1.6 (DMA ACKNOWLEDGE)**

When the DMA feature is implemented as described in the MCS® BASIC-52 users manual, this pin functions as an active low DMA ACKNOWLEDGE output.

**PORT 1.7 (LINE PRINTER OUTPUT)**

This pin functions as a serial output port when the LIST# or PRINT# command and/or statement is used. This enables the user to make a "hard copy" of a program or to print out results of a calculation.

**RESET**

A high (2.5 volts) on this pin for two machine cycles while the oscillator is running resets the device. An external pulldown resistor (~8.2K) from RESET to VSS permits power-on reset when a capacitor (~10 uf) is connected from this pin to VCC.

**ALE**

ALE (address latch enable) is an output pin that is used to latch the low order address byte during Read, Write, or program fetch operations to external memory.

**PSEN**

This pin (Program Store ENable) is a control signal that is used to enable external program memory. In MCS® BASIC-52, this pin will always remain inactive (logically high (1)) unless the user is running an assembly language program in external memory.

**XTAL1**

Input to the inverting amplifier that forms the oscillator.

**XTAL2**

Output of the inverting amplifier that forms the oscillator, and input to the internal clock generator. Receives the external oscillator signal when an external oscillator is used.

**RD**

A control signal that is used to enable READ operations to external data memory. This pin is active low (0).

**WR**

A control signal that is used to enable WRITE operations to external data memory. This pin is active low (0).

**T1**

This pin can be programmed to be an external input to TIMER/COUNTER 1.

**T0**

This pin can be programmed to be an external input to TIMER/COUNTER 0.

**INT1**

This pin is the external interrupt 1 pin. It is active low and interrupts on this pin may be handled in either BASIC or in assembly language.

**INT0/DMA REQUEST**

This is the external interrupt 0 pin. It is active low and may be optionally programmed to function as a DMA request input pin. The DMA REQUEST pin is used by E<sup>2</sup>PROM devices during programming.

**CONSOLE SERIAL OUTPUT**

This is the serial output pin to the console device. Standard ASCII codes are used as well as a standard asynchronous frame.

**CONSOLE SERIAL INPUT**

This is the serial input pin that receives data from the console device. Standard ASCII codes are assumed to be the input and the data is assumed to be transmitted using a standard asynchronous frame.

**NOTES**

If pin 31 is grounded the 8052AH-BASIC will operate as a standard 8032AH. The tolerances on this pin are described under DC characteristics.

For detailed information concerning this product please refer to the MCS BASIC-52 Users Manual (Order Number 210918-002).

**ABSOLUTE MAXIMUM RATINGS\***

Ambient Temperature Under Bias . . . 0°C to 70°C  
Storage Temperature . . . . . -65°C to +150°C  
Voltage on Any Pin With  
Respect to Ground (V<sub>SS</sub>) . . . . -0.5V to +7V  
Power Dissipation . . . . . 2 Watts

*\*NOTICE: Stresses above those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.*

**DC CHARACTERISTICS** (T<sub>A</sub> = 0°C to 70°C, V<sub>CC</sub> = 4.5V to 5.5V, V<sub>SS</sub> = 0V)

Symbol	Parameter	Min	Max	Unit	Test Conditions
VIL	Input Low Voltage	-0.5	0.8	V	
VIH	Input High Voltage (Except RST and XTAL2)	2.0	V <sub>CC</sub> + 0.5	V	
VIH1	Input High Voltage to RST for Reset, XTAL2	2.5	V <sub>CC</sub> + 0.5	V	XTAL1 to V <sub>SS</sub>
VOL	Output Low Voltage Port 1, A8-15, Control Functions		0.45	V	IOL = 1.6mA
VOL1	Output Low Voltage ALE, PSEN (Note 1)		0.45	V	IOL = 3.2mA
VOH	Output High Voltage Port 1, A8-15, Control Functions	2.4		V	IOH = -80μA
VOH1	Output High Voltage AD0-7, ALE, PSEN	2.4		V	IOH = -400μA
IIL	Logical 0 Input Current Port 1, A8-15 Control Functions		-800	μA	Vin = 0.45V
IIL2	Logical 0 Input Current XTAL2		-2.5	mA	XTAL1 at V <sub>SS</sub> , Vin = 0.45V
IL1	Input Leakage Current To AD0-7 EA		± 10	μA	0.45V < Vin < V <sub>CC</sub>
IIH1	Input High Current to RST/VPD For Reset		500	μA	Vin = V <sub>CC</sub> - 1.5V
ICC	Power Supply Current		175	mA	All outputs disconnected
CIO	Capacitance of I/O Buffer		10	pF	f <sub>c</sub> = 1MHz, T <sub>A</sub> = 25°C

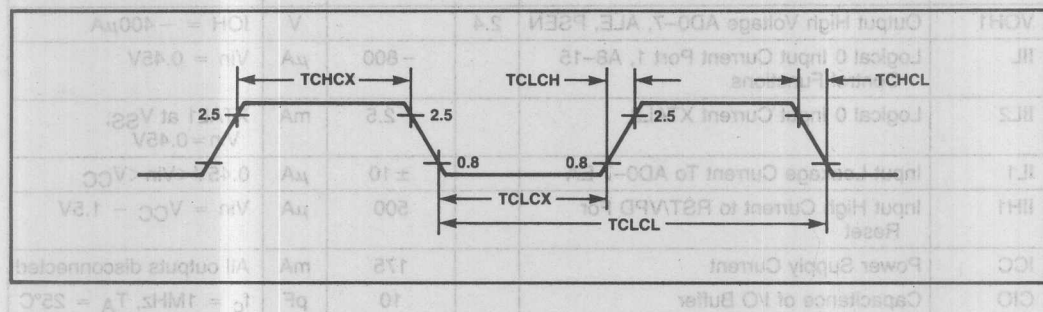
See page 6 for Notes.

**Note 1:** Vol is degraded when the 8032AH/8052AH rapidly discharges external capacitance. This AC noise is most pronounced during emission of address data. When using external memory, locate the latch or buffer as close to the 8032AH/8052AH as possible.

Datum	Emitting Ports	Degraded I/O Lines	VOL (peak) (max)
Address	A8-15, AD0-7	P1, Control Functions	0.8V
Write Data	AD0-7	P1, Control Functions, ALE	0.8v

### EXTERNAL CLOCK DRIVE CHARACTERISTICS (XTAL2)

Symbol	Parameter	Variable Clock $f = 3.5 \text{ MHz to } 12 \text{ MHz}$		Unit
		Min	Max	
TCLCL	Oscillator Period	83.3	286	ns
TCHCX	High Time	20		ns
TCLCX	Low Time	20		ns
TCLCH	Rise Time		20	ns
TCHCL	Fall Time		20	ns



**AC CHARACTERISTICS**  $T_A = 0^\circ\text{C to } 70^\circ\text{C}$ ,  $V_{CC} = 5V \pm 10\%$ ,  $V_{SS} = 0V$ ,  $C_L$  for AD0-7, ALE and  $\overline{PSEN}$  Outputs = 100 pF,  $C_L$  for all other outputs = 80 pF)

**PROGRAM MEMORY CHARACTERISTICS**

Symbol	Parameter	12 MHz Clock			Variable Clock 1/TCLCL = 3.5 MHz to 12 MHz		
		Min	Max	Unit	Min	Max	Unit
TLHLL	ALE Pulse Width	127		ns	2TCLCL-40		ns
TAVLL	Address Setup to ALE	43		ns	TCLCL-40		ns
TLLAX	Address Hold After ALE	48		ns	TCLCL-35		ns
TLLIV	ALE to Valid Instr In		233	ns		4TCLCL-100	ns
TLLPL	ALE To $\overline{PSEN}$	58		ns	TCLCL-25		ns
TPLPH	$\overline{PSEN}$ Pulse Width	215		ns	3TCLCL-35		ns
TPLIV	$\overline{PSEN}$ To Valid Instr In		125	ns		3TCLCL-125	ns
TPXIX	Input Instr Hold After $\overline{PSEN}$	0		ns	0		ns
TPXIZ	Input Instr Float After $\overline{PSEN}$		63	ns		TCLCL-20	ns
TPXAV	Address Valid After $\overline{PSEN}$	75		ns	TCLCL-8		ns
TAVIV	Address To Valid Instr In		302	ns		5TCLCL-115	ns
TAZPL	Address Float To $\overline{PSEN}$	0		ns	0		ns

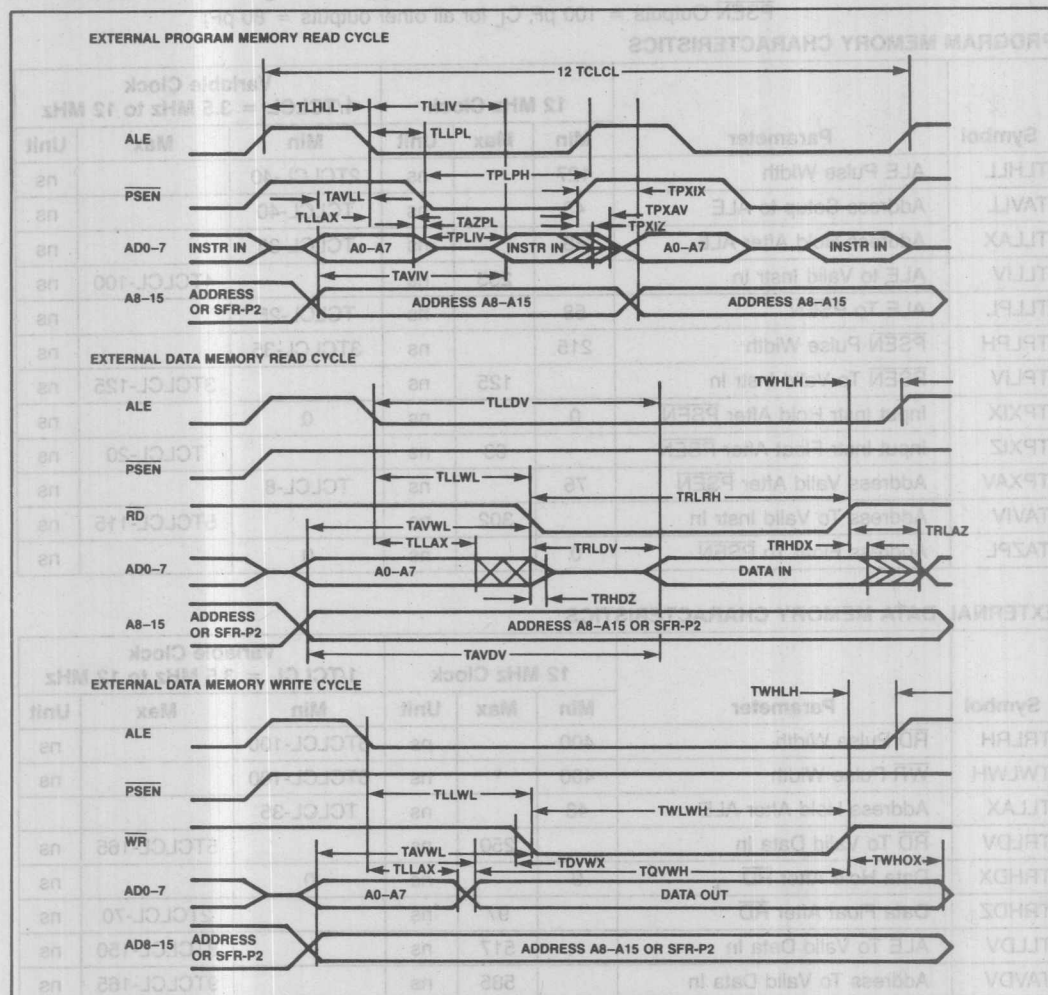
**EXTERNAL DATA MEMORY CHARACTERISTICS**

Symbol	Parameter	12 MHz Clock			Variable Clock 1/TCLCL = 3.5 MHz to 12 MHz		
		Min	Max	Unit	Min	Max	Unit
TRLRH	$\overline{RD}$ Pulse Width	400		ns	6TCLCL-100		ns
TWLWH	$\overline{WR}$ Pulse Width	400		ns	6TCLCL-100		ns
TLLAX	Address Hold After ALE	48		ns	TCLCL-35		ns
TRLDV	$\overline{RD}$ To Valid Data In		250	ns		5TCLCL-165	ns
TRHDX	Data Hold After $\overline{RD}$	0		ns	0		ns
TRHDZ	Data Float After $\overline{RD}$		97	ns		2TCLCL-70	ns
TLLDV	ALE To Valid Data In		517	ns		8TCLCL-150	ns
TAVDV	Address To Valid Data In		585	ns		9TCLCL-165	ns
TLLWL	ALE To $\overline{WR}$ or $\overline{RD}$	200	300	ns	3TCLCL-50	3TCLCL + 50	ns
TAVWL	Address To $\overline{SR}$ or $\overline{RD}$	203		ns	4TCLCL-130		ns
TWHLH	$\overline{WR}$ or $\overline{RD}$ High To ALE High	43	123	ns	TCLCL-40	TCLCL + 40	ns
TDVWX	Data Valid To $\overline{WR}$ Transition	23		ns	TCLCL-60		ns
TQVWH	Data Setup Before $\overline{WR}$	433		ns	7TCLCL-150		ns
TWHQX	Data Hold After $\overline{WR}$	33		ns	TCLCL-50		ns
TRLAZ	Address Float After $\overline{RD}$		0	ns		0	ns

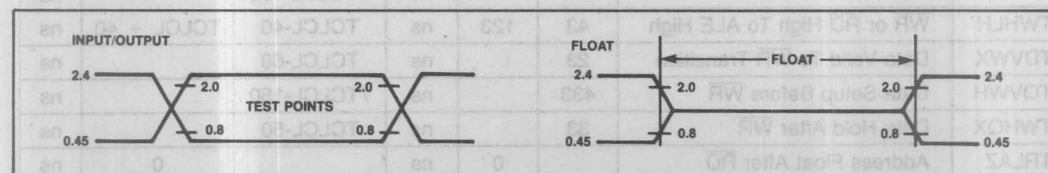
VO.5 is shown as an example. Timing measurements are taken with a logic analyzer or oscilloscope. The test signal is a square wave with a period of 10 ns and a duty cycle of 50%. The test signal is applied to the input of the device. The output of the device is measured. The test signal is applied to the input of the device. The output of the device is measured. The test signal is applied to the input of the device. The output of the device is measured.



# AC TIMING DIAGRAMS

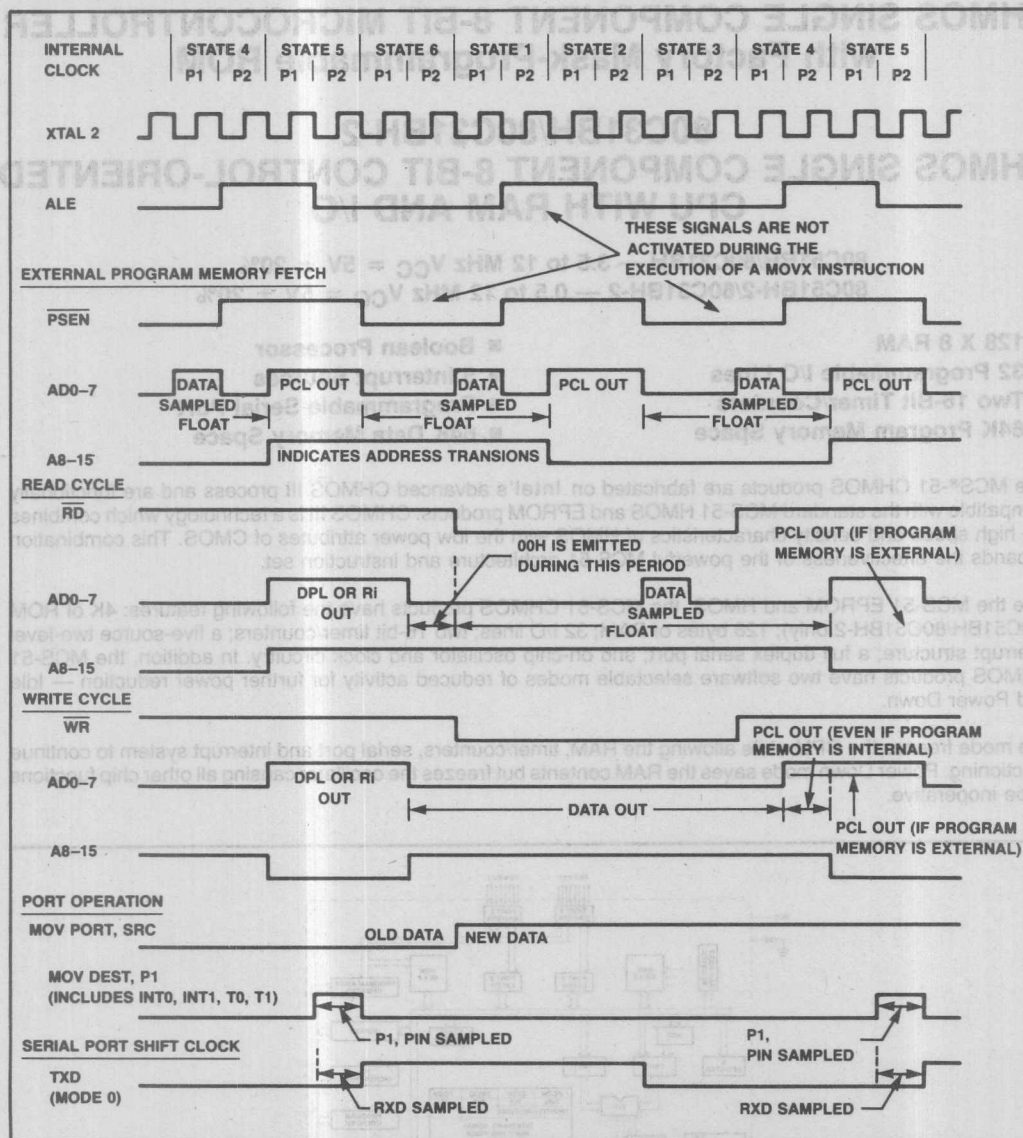


## AC TESTING INPUT/OUTPUT, FLOAT WAVEFORMS



AC inputs during testing are driven at 2.4V for a logic "1" and 0.45V for a logic "0". Timing measurements are made at 2.0V for a logic "1" and 0.8V for a logic "0". For timing purposes, the float state is defined as the point at which an AD0-7 pin sinks 2.4mA or sources 400 $\mu$ A at the voltage test levels.

# CLOCK WAVEFORMS



This diagram indicates when signals are clocked internally. The time it takes the signals to propagate to the pins, however, ranges from 25 to 125 ns. This propagation delay is dependent on variables such as temperature and pin loading. Propagation also varies from output to output and component to component. Typically though, ( $T_A = 25^\circ\text{C}$ , fully loaded) RD and WR propagation delays are approximately 50 ns. The other signals are typically 85 ns. Propagation delays are incorporated in the AC specifications.

# 80C51BH/80C51BH-2 CHMOS SINGLE COMPONENT 8-BIT MICROCONTROLLER with Factory Mask-Programmable ROM

## 80C31BH/80C31BH-2 CHMOS SINGLE COMPONENT 8-BIT CONTROL-ORIENTED CPU WITH RAM AND I/O

80C51BH/80C31BH — 3.5 to 12 MHz  $V_{CC} = 5V \pm 20\%$   
80C51BH-2/80C31BH-2 — 0.5 to 12 MHz  $V_{CC} = 5V \pm 20\%$

- 128 X 8 RAM
- 32 Programmable I/O Lines
- Two 16-Bit Timer/Counters
- 64K Program Memory Space
- Boolean Processor
- 5 Interrupt Sources
- Programmable Serial Port
- 64K Data Memory Space

The MCS®-51 CHMOS products are fabricated on Intel's advanced CHMOS III process and are functionally compatible with the standard MCS-51 HMOS and EPROM products. CHMOS III is a technology which combines the high speed and density characteristics of HMOS with the low power attributes of CMOS. This combination expands the effectiveness of the powerful MCS-51 architecture and instruction set.

Like the MCS-51 EPROM and HMOS, the MCS-51 CHMOS products have the following features: 4K of ROM (80C51BH/80C51BH-2 only); 128 bytes of RAM; 32 I/O lines; two 16-bit timer/counters; a five-source two-level interrupt structure; a full duplex serial port; and on-chip oscillator and clock circuitry. In addition, the MCS-51 CHMOS products have two software selectable modes of reduced activity for further power reduction — Idle and Power Down.

Idle mode freezes the CPU while allowing the RAM, timer/counters, serial port and interrupt system to continue functioning. Power Down mode saves the RAM contents but freezes the oscillator causing all other chip functions to be inoperative.

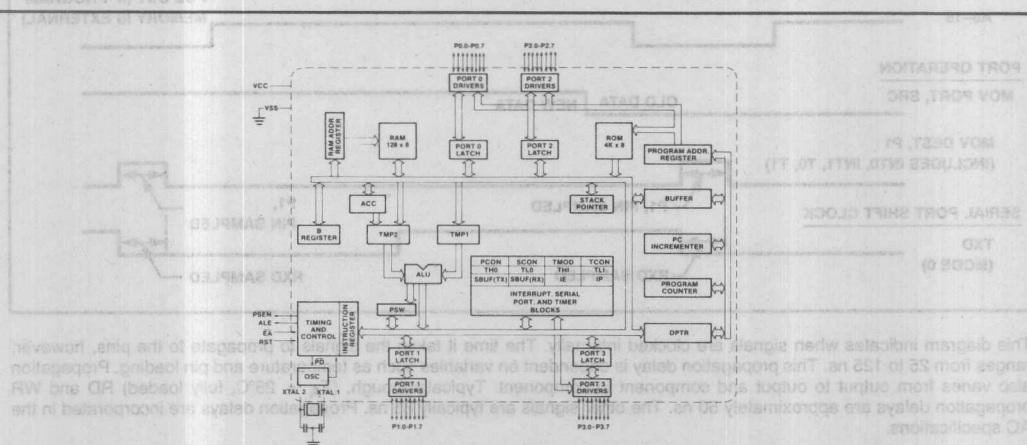


Figure 1. Block Diagram

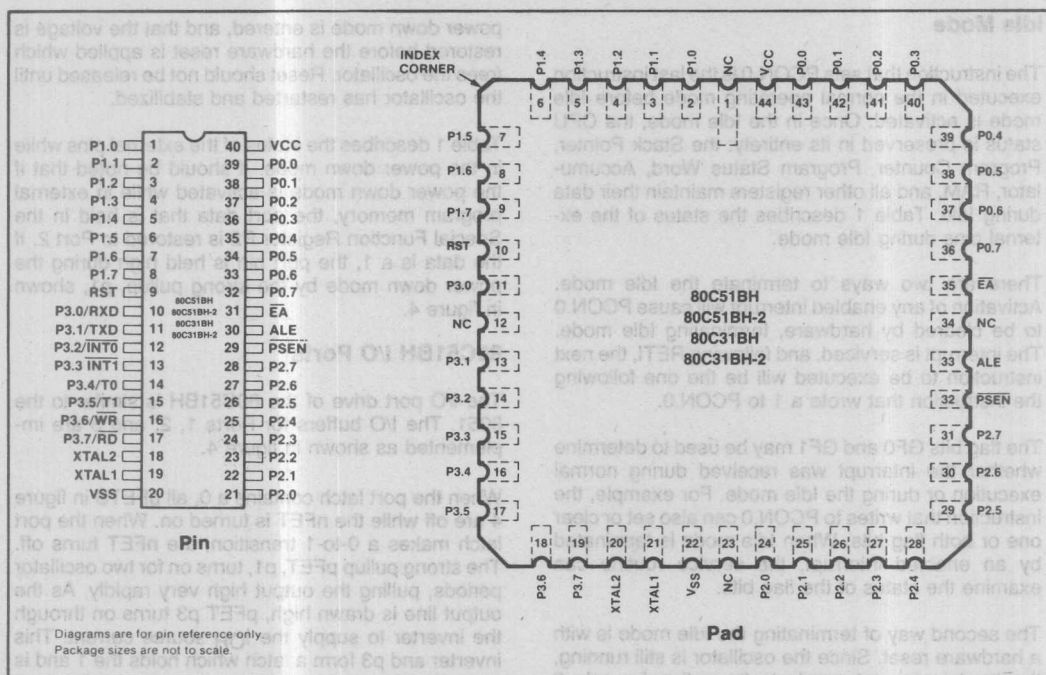


Figure 2. Configurations

## IDLE AND POWER DOWN OPERATION

Figure 3 shows the internal Idle and Power Down clock configuration. As illustrated, Power Down operation freezes the oscillator. Idle mode operation allows the interrupt, serial port, and timer blocks to continue to function while the clock to the CPU is halted.

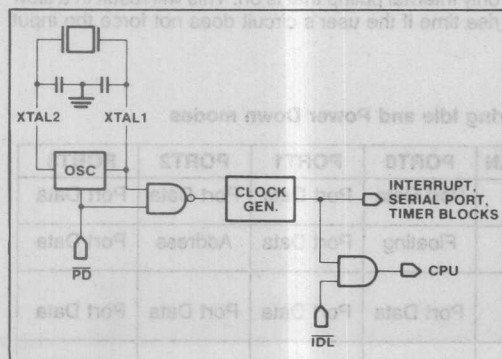


Figure 3. Idle and Power Down Hardware

These special modes are activated by software via the Special Function Register, PCON. Its hardware address is 87H. PCON is not bit addressable.

PCON: Power Control Register

(MSB)					(LSB)
SMOD	—	—	—	GF1	GF0
				PD	IDL

Symbol Position Name and Function

SMOD	PCON.7	Double Baud rate bit. When set to a 1, the baud rate is doubled when the serial port is being used in either modes 1, 2 or 3.
—	PCON.6	(Reserved)
—	PCON.5	(Reserved)
—	PCON.4	(Reserved)
GF1	PCON.3	General-purpose flag bit.
GF0	PCON.2	General-purpose flag bit.
PD	PCON.1	Power Down bit. Setting this bit activates power down operation.
IDL	PCON.0	Idle mode bit. Setting this bit activates idle mode operation.

If 1's are written to PD and IDL at the same time, PD takes precedence. The reset value of PCON is (0XXX0000).



## Idle Mode

The instruction that sets PCON.0 is the last instruction executed in the normal operating mode before Idle mode is activated. Once in the Idle mode, the CPU status is preserved in its entirety: the Stack Pointer, Program Counter, Program Status Word, Accumulator, RAM, and all other registers maintain their data during Idle. Table 1 describes the status of the external pins during Idle mode.

There are two ways to terminate the Idle mode. Activation of any enabled interrupt will cause PCON.0 to be cleared by hardware, terminating Idle mode. The interrupt is serviced, and following RETI, the next instruction to be executed will be the one following the instruction that wrote a 1 to PCON.0.

The flag bits GF0 and GF1 may be used to determine whether the interrupt was received during normal execution or during the Idle mode. For example, the instruction that writes to PCON.0 can also set or clear one or both flag bits. When Idle mode is terminated by an enabled interrupt, the service routine can examine the status of the flag bits.

The second way of terminating the Idle mode is with a hardware reset. Since the oscillator is still running, the hardware reset needs to be active for only 2 machine cycles (24 oscillator periods) to complete the reset operation.

## Power Down Mode

The instruction that sets PCON.1 is the last executed prior to going into power down. Once in power down, the oscillator is stopped. Only the contents of the on-chip RAM is preserved. The Special Function Registers are not saved. A hardware reset is the only way of exiting the power down mode.

In the Power Down mode,  $V_{CC}$  may be lowered to minimize circuit power consumption. Care must be taken to ensure the voltage is not reduced until the

power down mode is entered, and that the voltage is restored before the hardware reset is applied which frees the oscillator. Reset should not be released until the oscillator has restarted and stabilized.

Table 1 describes the status of the external pins while in the power down mode. It should be noted that if the power down mode is activated while in external program memory, the port data that is held in the Special Function Register P2 is restored to Port 2. If the data is a 1, the port pin is held high during the power down mode by the strong pullup, p1, shown in figure 4.

## 80C51BH I/O Ports

The I/O port drive of the 80C51BH is similar to the 8051. The I/O buffers for Ports 1, 2, and 3 are implemented as shown in figure 4.

When the port latch contains a 0, all pFETs in figure 4 are off while the nFET is turned on. When the port latch makes a 0-to-1 transition, the nFET turns off. The strong pullup pFET, p1, turns on for two oscillator periods, pulling the output high very rapidly. As the output line is drawn high, pFET p3 turns on through the inverter to supply the  $I_{OH}$  source current. This inverter and p3 form a latch which holds the 1 and is supported by p2.

When Port 2 is used as an address port, for access to external program of data memory, any address bit that contains a 1 will have its strong pullup turned on for the entire duration of the external memory access.

When an I/O pin on Ports 1, 2, or 3 is used as an input, the user should be aware that the external circuit must sink current during the logical 1-to-0 transition. The maximum sink current is specified as I<sub>TL</sub> under the D.C. Specifications. When the input goes below approximately 2V, p3 turns off to save ICC current. Note, when returning to a logical 1, p2 is the only internal pullup that is on. This will result in a slow rise time if the user's circuit does not force the input

Table 1. Status of the external pins during Idle and Power Down modes

Mode	Program Memory	ALE	PSEN	PORT0	PORT1	PORT2	PORT3
Idle	Internal	1	1	Port Data	Port Data	Port Data	Port Data
Idle	External	1	1	Floating	Port Data	Address	Port Data
Power Down	Internal	0	0	Port Data	Port Data	Port Data	Port Data
Power Down	External	0	0	Floating	Port Data	Port Data	Port Data

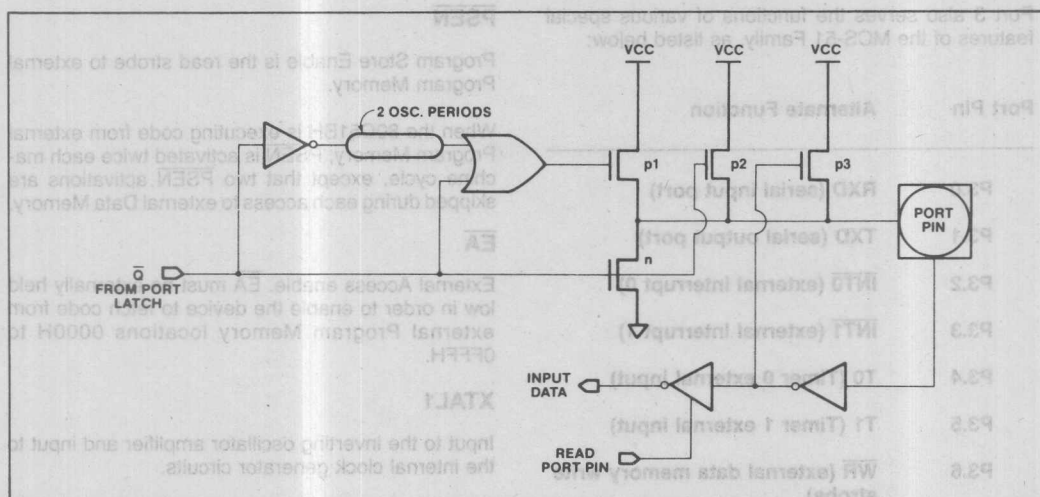


Figure 4. I/O Buffers in the 80C51BH (Ports 1, 2, 3)

line high. For additional information, refer to the chapter entitled "Design Considerations When Using CHMOS" in the 1984 Intel Microcontroller Handbook.

## PIN DESCRIPTIONS

### VCC

Supply voltage during normal, Idle, and Power Down operations.

### VSS

Circuit ground.

### Port 0

Port 0 is an 8-bit open drain bi-directional I/O port. Port 0 pins that have 1's written to them float, and in that state can be used as high-impedance inputs.

Port 0 is also the multiplexed low-order address and data bus during accesses to external Program and Data Memory. In this application it uses strong internal pullups when emitting 1's. Port 0 also outputs the code bytes during program verification in the 80C51BH. External pullups are required during program verification.

### Port 1

Port 1 is an 8-bit bi-directional I/O port with internal pullups. Port 1 pins that have 1's written to them are

pulled high by the internal pullups, and in that state can be used as inputs. As inputs, Port 1 pins that are externally being pulled low will source current (IIL, on the data sheet) because of the internal pullups.

Port 1 also receives the low-order address bytes during program verification.

### Port 2

Port 2 is an 8-bit bi-directional I/O port with internal pullups. Port 2 pins that have 1's written to them are pulled high by the internal pullups, and in that state can be used as inputs. As inputs, Port 2 pins that are externally being pulled low will source current (IIL, on the data sheet) because of the internal pullups.

Port 2 emits the high-order address byte during fetches from external Program Memory and during accesses to external Data Memory that use 16-bit addresses (MOVX @DPTR). In this application it uses strong internal pullups when emitting 1's. During accesses to external Data Memory that use 8-bit addresses (MOVX @Ri), Port 2 emits the contents of the P2 Special Function Register.

### Port 3

Port 3 is an 8-bit bi-directional I/O port with internal pullups. Port 3 pins that have 1's written to them are pulled high by the internal pullups, and in that state can be used as inputs. As inputs, Port 3 pins that are externally being pulled low will source current (IIL, on the data sheet) because of the pullups.

Port 3 also serves the functions of various special features of the MCS-51 Family, as listed below:

Port Pin	Alternate Function
P3.0	RXD (serial input port)
P3.1	TXD (serial output port)
P3.2	$\overline{\text{INT0}}$ (external interrupt 0)
P3.3	$\overline{\text{INT1}}$ (external interrupt 1)
P3.4	T0 (Timer 0 external input)
P3.5	T1 (Timer 1 external input)
P3.6	$\overline{\text{WR}}$ (external data memory write strobe)
P3.7	$\overline{\text{RD}}$ (external data memory read strobe)

## RST

Reset input. A high on this pin for two machine cycles while the oscillator is running resets the device. An internal diffused resistor to  $V_{SS}$  permits Power-On reset using only an external capacitor to  $V_{CC}$ .

## ALE

Address Latch Enable output pulse for latching the low byte of the address during accesses to external memory.

In normal operation ALE is emitted at a constant rate of  $\frac{1}{6}$  the oscillator frequency, and may be used for external timing or clocking purposes. Note, however, that one ALE pulse is skipped during each access to external Data Memory.

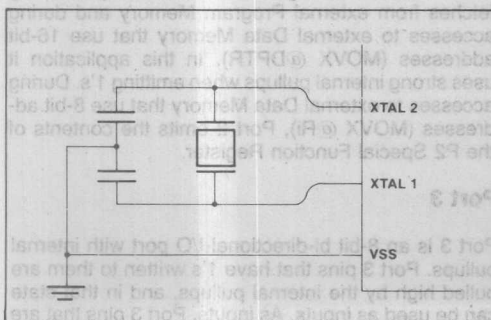


Figure 5. Crystal Oscillator

## PSEN

Program Store Enable is the read strobe to external Program Memory.

When the 80C51BH is executing code from external Program Memory,  $\overline{\text{PSEN}}$  is activated twice each machine cycle, except that two  $\overline{\text{PSEN}}$  activations are skipped during each access to external Data Memory.

## EA

External Access enable.  $\overline{\text{EA}}$  must be externally held low in order to enable the device to fetch code from external Program Memory locations 0000H to 0FFFH.

## XTAL1

Input to the inverting oscillator amplifier and input to the internal clock generator circuits.

## XTAL2

Output from the inverting oscillator amplifier.

## OSCILLATOR CHARACTERISTICS

XTAL1 and XTAL2 are the input and output respectively, of an inverting amplifier which is configured for use as an on-chip oscillator, as shown in Figure 5. Either a quartz crystal or ceramic resonator may be used. More detailed information concerning the use of the on-chip oscillator is available in Application Note AP-155, "Oscillators for Microcontrollers."

To drive the device from an external clock source, XTAL1 should be driven while XTAL2 is left unconnected as shown in figure 6. There are no requirements on the duty cycle of the external clock signal, since the input to the internal clocking circuitry is through a divide-by-two flip-flop, but minimum and maximum high and low times specified on the Data Sheet must be observed.

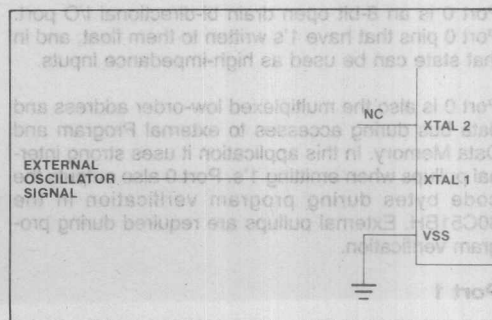


Figure 6. External Drive Configuration

# **ABSOLUTE MAXIMUM RATINGS\***

Ambient Temperature Under Bias . . . . . 0°C to 70°C  
Storage Temperature . . . . . -65°C to +150°C  
Voltage on Any Pin to V<sub>SS</sub> . . . . . -0.5V to V<sub>CC</sub>+1V  
Voltage on V<sub>CC</sub> to V<sub>SS</sub> . . . . . -0.5V to +7V  
Power Dissipation . . . . . 1.0W

\*NOTICE: Stresses above those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.

## **D.C. CHARACTERISTICS:** (T<sub>A</sub> = 0°C to 70°C; V<sub>SS</sub> = 0V; V<sub>CC</sub> = 5V ± 20%)

Symbol	Parameter	Min	Max	Unit	Test Conditions
V <sub>IL</sub>	Input Low Voltage	-0.5	0.2V <sub>CC</sub> - .1	V	
V <sub>IH</sub>	Input High Voltage (Except XTAL1, RST)	0.2V <sub>CC</sub> + .9	V <sub>CC</sub> + 0.5	V	
V <sub>IH1</sub>	Input High Voltage to XTAL1, RST	0.7V <sub>CC</sub>	V <sub>CC</sub> + 0.5	V	
V <sub>OL</sub>	Output Low Voltage (Ports 1, 2, 3)		0.45	V	I <sub>OL</sub> = 1.6 mA
V <sub>OL1</sub>	Output Low Voltage (Port 0, ALE, PSEN)		0.45	V	I <sub>OL</sub> = 3.2 mA (Note 1)
V <sub>OH</sub>	Output High Voltage (Ports 1, 2, 3)	2.4		V	I <sub>OH</sub> = -80μA V <sub>CC</sub> = 5V ± 10%
		0.75V <sub>CC</sub>		V	I <sub>OH</sub> = -30μA
		0.9V <sub>CC</sub>		V	I <sub>OH</sub> = -10μA
V <sub>OH1</sub>	Output High Voltage (Port 0 in External Bus Mode, ALE, PSEN)	2.4		V	I <sub>OH</sub> = -400μA V <sub>CC</sub> = 5V ± 10%
		0.75V <sub>CC</sub>		V	I <sub>OH</sub> = -150μA
		0.9V <sub>CC</sub>		V	I <sub>OH</sub> = -40μA (Note 2)
I <sub>IL</sub>	Logical 0 Input Current (Ports 1, 2, 3)		-50	μA	V <sub>in</sub> = 0.45V
I <sub>TL</sub>	Logical 1 to 0 transition Current (Ports 1, 2, 3)		-500	μA	V <sub>in</sub> = 2.0V
I <sub>LI</sub>	Input Leakage Current (Port 0, EA)		± 10	μA	0.45 < V <sub>in</sub> < V <sub>CC</sub>
R <sub>RST</sub>	RST Pulldown Resistor	40	125	Kohm	
C <sub>IO</sub>	Pin Capacitance		10	pF	test freq = 1 MHz, T <sub>A</sub> = 25°C
I <sub>PD</sub>	Power Down Current		50	μA	V <sub>CC</sub> = 2 to 6V (Note 3)

### **Maximum Operating I<sub>CC</sub> (mA) (note 4)**

V <sub>CC</sub>	4V	5V	6V
Freq.			
0.5 MHz	1.6	2.2	3
3.5 MHz	4.3	5.7	7.5
8 MHz	8.3	11	14
12 MHz	12	16	20

### **Maximum Idle I<sub>CC</sub> (mA) (note 5)**

V <sub>CC</sub>	4V	5V	6V
Freq.			
0.5 MHz	0.6	0.9	1.2
3.5 MHz	1.1	1.6	2.2
8 MHz	1.8	2.7	3.7
12 MHz	2.5	3.7	5



**Note 1:** Capacitive loading on Ports 0 and 2 may cause spurious noise pulses to be superimposed on the  $V_{OL}$ s of ALE, and Ports 1 and 3. The noise is due to the external bus capacitance discharging into the Port 0 and Port 2 pins when these pins make a 1-to-0 transition during bus operations. In the worst case (capacitive loading  $> 100$  pF), the noise pulse on ALE line may exceed 0.8V. In such cases it may be desirable to qualify ALE with a Schmitt Trigger, or use an address latch with a Schmitt Trigger STROBE input.

**Note 2:** Capacitive loading on Ports 0 and 2 may cause the  $V_{OH}$  on ALE and PSEN to momentarily fall below the  $.9V_{CC}$  specification when the address bits are stabilizing.

**Note 3:** Power Down  $I_{CC}$  is measured with all output pins disconnected; EA = PORT0 =  $V_{CC}$ ; XTAL2 N.C.; RST =  $V_{SS}$ .

**Note 4:**  $I_{CC}$  is measured with all output pins disconnected; XTAL1 driven with TCLCH, TCHCL = 10 ns,  $V_{II} = V_{SS} + .5v$ ,  $V_{IH} = V_{CC} - .5v$ ; XTAL2 N.C.; EA = RST = PORT0 =  $V_{CC}$ .

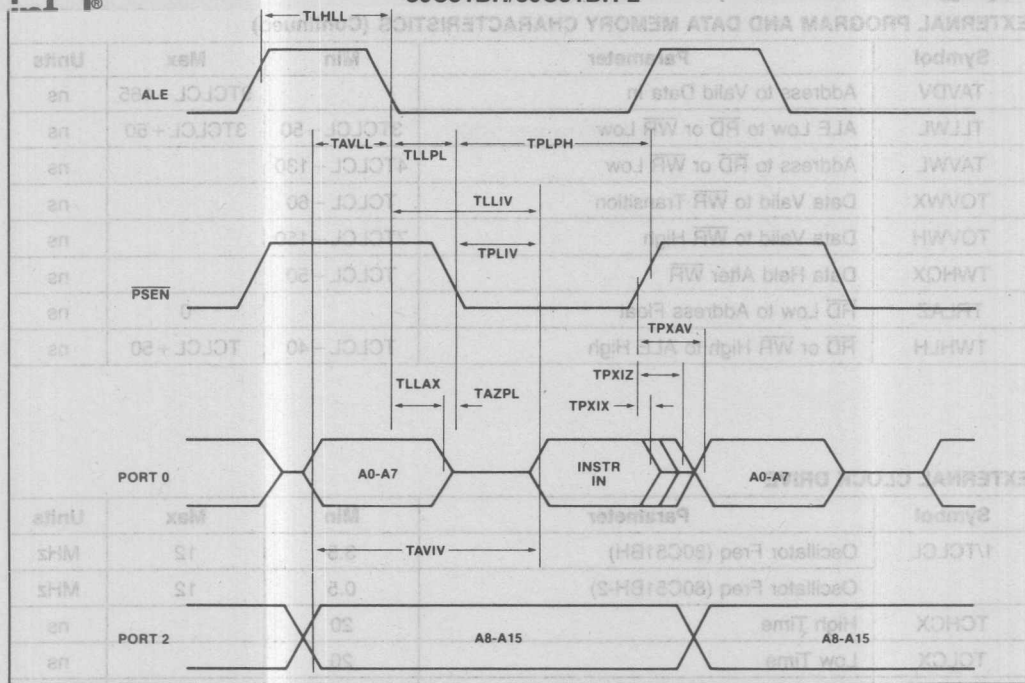
**Note 5:** Idle  $I_{CC}$  is measured with all output pins disconnected; XTAL1 driven with TCLCH, TCHCL = 10 ns,  $V_{II} = V_{SS} + .5v$ ,  $V_{IH} = V_{CC} - .5v$ ; XTAL2 N.C.; EA = PORT0 =  $V_{CC}$ ; RST =  $V_{SS}$ .

**A.C. CHARACTERISTICS** ( $T_A = 0^\circ\text{C}$  to  $70^\circ\text{C}$ ;  $V_{SS} = 0V$ ;  $V_{CC} = 5V \pm 20\%$ ;  
Load Capacitance for Port 0, ALE, and PSEN = 100 pF, Load Capacitance for  
All Other Outputs = 80 pF)

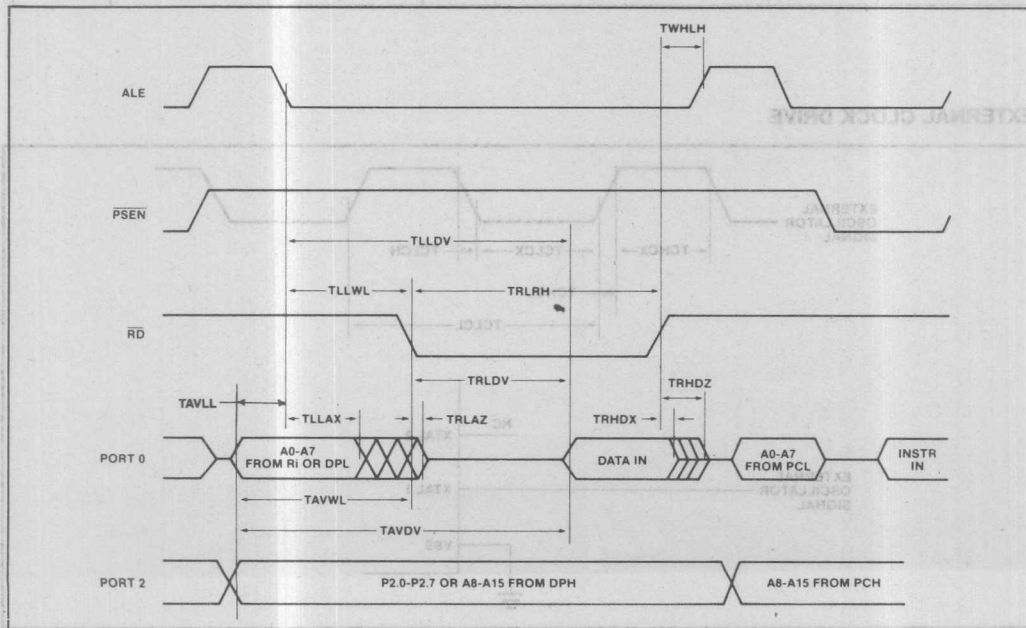
#### EXTERNAL PROGRAM AND DATA MEMORY CHARACTERISTICS

Symbol	Parameter	Min	Max	Units
1/TCLCL	Oscillator Freq (80C51BH)	3.5	12	MHz
	Oscillator Freq (80C51BH-2)	0.5	12	MHz
TLHLL	ALE Pulse Width	2TCLCL - 40		ns
TAVLL	Address Valid to ALE Low	TCLCL - 40		ns
TLLAX	Address Hold After ALE Low	TCLCL - 35		ns
TLLIV	ALE Low to Valid Instr In		4TCLCL - 150	ns
TLLPL	ALE Low to PSEN Low	TCLCL - 25		ns
TPLPH	PSEN Pulse Width	3TCLCL - 35		ns
TPLIV	PSEN Low to Valid Instr In		3TCLCL - 150	ns
TPXIX	Input Instr Hold After PSEN	0		ns
TPXIZ	Input Instr Float After PSEN		TCLCL - 20	ns
TPXAV	PSEN to Address Valid	TCLCL - 8		ns
TAVIV	Address to Valid Instr In		5TCLCL - 150	ns
TPLAZ	PSEN Low to Address Float		0	ns
TRLRH	RD Pulse Width	6TCLCL - 100		ns
TWLWH	WR Pulse Width	6TCLCL - 100		ns
TRLDV	RD Low to Valid Data In		5TCLCL - 165	ns
TRHDX	Data Hold After RD	0		ns
TRHDZ	Data Float After RD		2TCLCL - 70	ns
TLLDV	ALE Low to Valid Data In		8TCLCL - 150	ns

# 80C51BH/80C51BH-2



## EXTERNAL DATA MEMORY READ CYCLE



# A.C. CHARACTERISTICS

EXTERNAL PROGRAM MEMORY READ CYCLE

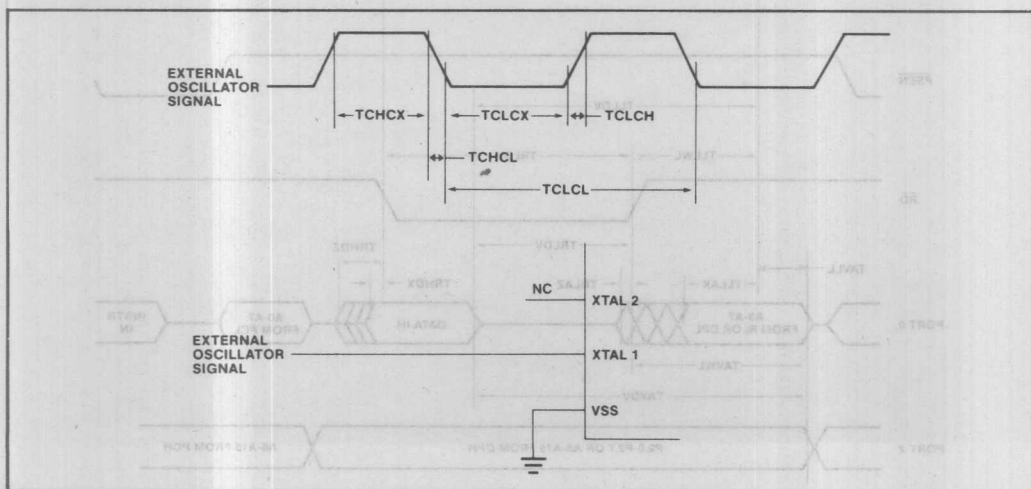
## EXTERNAL PROGRAM AND DATA MEMORY CHARACTERISTICS (Continued)

Symbol	Parameter	Min	Max	Units
TAVDV	Address to Valid Data In		9TCLCL - 165	ns
TLLWL	ALE Low to $\overline{RD}$ or $\overline{WR}$ Low	3TCLCL - 50	3TCLCL + 50	ns
TAVWL	Address to $\overline{RD}$ or $\overline{WR}$ Low	4TCLCL - 130		ns
TQVWX	Data Valid to $\overline{WR}$ Transition	TCLCL - 60		ns
TQVWH	Data Valid to $\overline{WR}$ High	7TCLCL - 150		ns
TWHQX	Data Held After $\overline{WR}$	TCLCL - 50		ns
TRLAZ	$\overline{RD}$ Low to Address Float		0	ns
TWHLH	$\overline{RD}$ or $\overline{WR}$ High to ALE High	TCLCL - 40	TCLCL + 50	ns

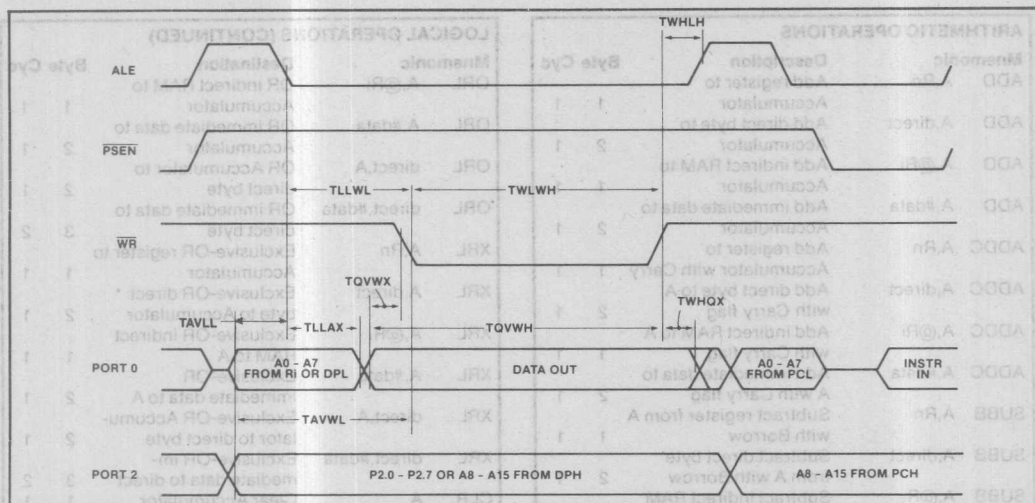
## EXTERNAL CLOCK DRIVE

Symbol	Parameter	Min	Max	Units
1/TCLCL	Oscillator Freq (80C51BH)	3.5	12	MHz
	Oscillator Freq (80C51BH-2)	0.5	12	MHz
TCHCX	High Time	20		ns
TCLCX	Low Time	20		ns
TCLCH	Rise Time		20	ns
TCHCL	Fall Time		20	ns

## EXTERNAL CLOCK DRIVE



EXTERNAL DATA MEMORY WRITE CYCLE

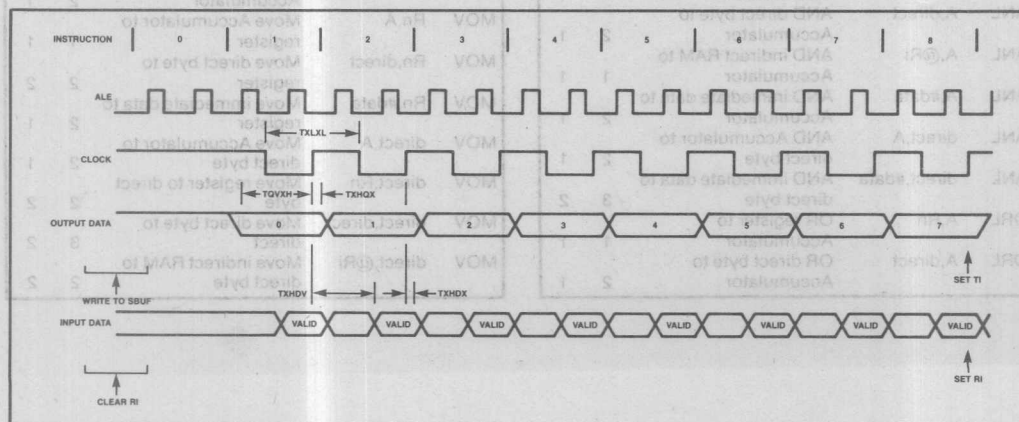


### SERIAL PORT TIMING — SHIFT REGISTER MODE

A.C. CHARACTERISTICS: ( $T_A = 0^\circ\text{C}$  to  $70^\circ\text{C}$ ;  $V_{SS} = 0\text{V}$ ;  $V_{CC} = 5\text{V} \pm 20\%$ ;  
Load Capacitance =  $80\text{ pF}$ )

Symbol	Parameter	Min	Max	Units
TXLXL	Serial Port Clock Cycle Time	12TCLCL		$\mu\text{s}$
TQVXH	Output Data Setup to Clock Rising Edge	10TCLCL - 133		ns
TXHQX	Output Data Hold After Clock Rising Edge	2TCLCL - 117		ns
TXHDX	Input Data Hold After Clock Rising Edge	0		ns
TXHDV	Clock Rising Edge to Input Data Valid		10TCLCL - 133	ns

### SHIFT REGISTER TIMING WAVEFORMS





**Table 2. MCS®-51 Instruction Set Description**

ARITHMETIC OPERATIONS			LOGICAL OPERATIONS (CONTINUED)		
Mnemonic		Description	Byte	Cyc	
ADD	A,Rn	Add register to Accumulator	1	1	
ADD	A,direct	Add direct byte to Accumulator	2	1	
ADD	A,@Ri	Add indirect RAM to Accumulator	1	1	
ADD	A,#data	Add immediate data to Accumulator	2	1	
ADDC	A,Rn	Add register to Accumulator with Carry	1	1	
ADDC	A,direct	Add direct byte to A with Carry flag	2	1	
ADDC	A,@Ri	Add indirect RAM to A with Carry flag	1	1	
ADDC	A,#data	Add immediate data to A with Carry flag	2	1	
SUBB	A,Rn	Subtract register from A with Borrow	1	1	
SUBB	A,direct	Subtract direct byte from A with Borrow	2	1	
SUBB	A,@Ri	Subtract indirect RAM from A with Borrow	1	1	
SUBB	A,#data	Subtract immed data from A with Borrow	2	1	
INC	A	Increment Accumulator	1	1	
INC	Rn	Increment register	1	1	
INC	direct	Increment direct byte	2	1	
INC	@Ri	Increment indirect RAM	1	1	
INC	DPTR	Increment Data Pointer	1	2	
DEC	A	Decrement Accumulator	1	1	
DEC	Rn	Decrement register	1	1	
DEC	direct	Decrement direct byte	2	1	
DEC	@Ri	Decrement indirect RAM	1	1	
MUL	AB	Multiply A & B	1	4	
DIV	AB	Divide A by B	1	4	
DA	A	Decimal Adjust Accumulator	1	1	
LOGICAL OPERATIONS			DATA TRANSFER		
Mnemonic		Destination	Byte	Cyc	
ANL	A,Rn	AND register to Accumulator	1	1	
ANL	A,direct	AND direct byte to Accumulator	2	1	
ANL	A,@Ri	AND indirect RAM to Accumulator	1	1	
ANL	A,#data	AND immediate data to Accumulator	2	1	
ANL	direct,A	AND Accumulator to direct byte	2	1	
ANL	direct,#data	AND immediate data to direct byte	3	2	
ORL	A,Rn	OR register to Accumulator	1	1	
ORL	A,direct	OR direct byte to Accumulator	2	1	
LOGICAL OPERATIONS (CONTINUED)			DATA TRANSFER		
Mnemonic		Destination	Byte	Cyc	
ORL	A,@Ri	OR indirect RAM to Accumulator	1	1	
ORL	A,#data	OR immediate data to Accumulator	2	1	
ORL	direct,A	OR Accumulator to direct byte	2	1	
ORL	direct,#data	OR immediate data to direct byte	3	2	
XRL	A,Rn	Exclusive-OR register to Accumulator	1	1	
XRL	A,direct	Exclusive-OR direct byte to Accumulator	2	1	
XRL	A,@Ri	Exclusive-OR indirect RAM to A	1	1	
XRL	A,#data	Exclusive-OR immediate data to A	2	1	
XRL	direct,A	Exclusive-OR Accumulator to direct byte	2	1	
XRL	direct,#data	Exclusive-OR immediate data to direct	3	2	
CLR	A	Clear Accumulator	1	1	
CPL	A	Complement Accumulator	1	1	
RL	A	Rotate Accumulator Left	1	1	
RLC	A	Rotate A Left through the Carry flag	1	1	
RR	A	Rotate Accumulator Right	1	1	
RRC	A	Rotate A Right through Carry flag	1	1	
SWAP	A	Swap nibbles within the Accumulator	1	1	
DATA TRANSFER			DATA TRANSFER		
Mnemonic		Description	Byte	Cyc	
MOV	A,Rn	Move register to Accumulator	1	1	
MOV	A,direct	Move direct byte to Accumulator	2	1	
MOV	A,@Ri	Move indirect RAM to Accumulator	1	1	
MOV	A,#data	Move immediate data to Accumulator	2	1	
MOV	Rn,A	Move Accumulator to register	1	1	
MOV	Rn,direct	Move direct byte to register	2	2	
MOV	Rn,#data	Move immediate data to register	2	1	
MOV	direct,A	Move Accumulator to direct byte	2	1	
MOV	direct,Rn	Move register to direct byte	2	2	
MOV	direct,direct	Move direct byte to direct	3	2	
MOV	direct,@Ri	Move indirect RAM to direct byte	2	2	

Table 2. MCS<sup>®</sup>-51 Instruction Set Description (Continued)

DATA TRANSFER (CONTINUED)			PROGRAM AND MACHINE CONTROL		
Mnemonic	Description	Byte Cyc	Mnemonic	Description	Byte Cyc
MOV direct,#data	Move immediate data to direct byte	3 2	ACALL addr11	Absolute Subroutine Call	2 2
MOV @Ri,A	Move Accumulator to indirect RAM	1 1	LCALL addr16	Long Subroutine Call	3 2
MOV @Ri,direct	Move direct byte to indirect RAM	2 2	RET	Return from subroutine	1 2
MOV @Ri,#data	Move immediate data to indirect RAM	2 1	RETI	Return from interrupt	1 2
MOV DPTR,#data16	Load Data Pointer with a 16-bit constant	3 2	AJMP addr11	Absolute Jump	2 2
MOVC A,@A+DPTR	Move Code byte relative to DPTR to A	1 2	LJMP addr16	Long Jump	3 2
MOVC A,@A+PC	Move Code byte relative to PC to A	1 2	SJMP rel	Short Jump (relative addr)	2 2
MOVB A,@Ri	Move External RAM (8-bit addr) to A	1 2	JMP @A+DPTR	Jump indirect relative to the DPTR	1 2
MOVB A,@DPTR	Move External RAM (16-bit addr) to A	1 2	JZ rel	Jump if Accumulator is Zero	2 2
MOVB @Ri,A	Move A to External RAM (8-bit addr)	1 2	JNZ rel	Jump if Accumulator is Not Zero	2 2
MOVB @DPTR,A	Move A to External RAM (16-bit addr)	1 2	JC rel	Jump if Carry flag is set	2 2
PUSH direct	Push direct byte onto stack	2 2	JNC rel	Jump if No Carry flag	2 2
POP direct	Pop direct byte from stack	2 2	JB bit,rel	Jump if direct Bit set	3 2
XCH A,Rn	Exchange register with Accumulator	1 1	JNB bit,rel	Jump if direct Bit Not set	3 2
XCH A,direct	Exchange direct byte with Accumulator	2 1	JBC bit,rel	Jump if direct Bit is set & Clear bit	3 2
XCH A,@Ri	Exchange indirect RAM with A	1 1	CJNE A,direct,rel	Compare direct to A & Jump if Not Equal	3 2
XCHD A,@Ri	Exchange low-order Digit ind RAM w A	1 1	CJNE A,#data,rel	Comp, immed, to A & Jump if Not Equal	3 2
BOOLEAN VARIABLE MANIPULATION			CJNE Rn,#data,rel	Comp, immed, to reg & Jump if Not Equal	3 2
Mnemonic	Description	Byte Cyc	CJNE @Ri,#data,rel	Comp, immed, to ind, & Jump if Not Equal	3 2
CLR C	Clear Carry flag	1 1	DJNZ Rn,rel	Decrement register & Jump if Not Zero	2 2
CLR bit	Clear direct bit	2 1	DJNZ direct,rel	Decrement direct & Jump if Not Zero	3 2
SETB C	Set Carry flag	1 1	NOP	No operation	1 1
SETB bit	Set direct Bit	2 1	Notes on data addressing modes:		
CPL C	Complement Carry flag	1 1	Rn	Working register R0-R7	
CPL bit	Complement direct bit	2 1	direct	128 internal RAM locations, any I/O port, control or status register	
ANL C,bit	AND direct bit to Carry flag	2 2	@Ri	Indirect internal RAM location addressed by register R0 or R1	
ANL C,/bit	AND complement of direct bit to Carry flag	2 2	#data	8-bit constant included in instruction	
ORL C,bit	OR direct bit to Carry flag	2 2	#data16	16-bit constant included as bytes 2 & 3 of instruction	
ORL C,/bit	OR complement of direct bit to Carry flag	2 2	bit	128 software flags, any I/O pin, control or status bit	
MOV C,/bit	Move direct bit to Carry flag	2 1	Notes on program addressing modes:		
MOV bit,C	Move Carry flag to direct bit	2 2	addr16	Destination address for LCALL & LJMP may be anywhere within the 64-K program memory address space	
			Addr11	Destination address for ACALL & AJMP will be within the same 2-K page of program memory as the first byte of the following instruction	
			rel	SJMP and all conditional jumps include an 8-bit offset byte. Range is +127-128 bytes relative to first byte of the following instruction	
			All mnemonics copyrighted © Intel Corporation 1979		

# 80C51BH/80C51BH-2

Hex Code	Number of Bytes	Mnemonic	Operands
00	1	NOP	
01	2	AJMP	code addr
02	3	LJMP	code addr
03	1	RR	A
04	1	INC	A
05	2	INC	data addr
06	1	INC	@R0
07	1	INC	@R1
08	1	INC	R0
09	1	INC	R1
0A	1	INC	R2
0B	1	INC	R3
0C	1	INC	R4
0D	1	INC	R5
0E	1	INC	R6
0F	1	INC	R7
10	3	JBC	bit addr, code addr
11	2	ACALL	code addr
12	3	LCALL	code addr
13	1	RRC	A
14	1	DEC	A
15	2	DEC	data addr
16	1	DEC	@R0
17	1	DEC	@R1
18	1	DEC	R0
19	1	DEC	R1
1A	1	DEC	R2
1B	1	DEC	R3
1C	1	DEC	R4
1D	1	DEC	R5
1E	1	DEC	R6
1F	1	DEC	R7
20	3	JB	bit addr, code addr
21	2	AJMP	code addr
22	1	RET	
23	1	RL	A
24	2	ADD	A, #data
25	2	ADD	A, data addr
26	1	ADD	A, @R0
27	1	ADD	A, @R1
28	1	ADD	A, R0
29	1	ADD	A, R1
2A	1	ADD	A, R2
2B	1	ADD	A, R3
2C	1	ADD	A, R4
2D	1	ADD	A, R5
2E	1	ADD	A, R6
2F	1	ADD	A, R7
30	3	JNB	bit addr, code addr
31	2	ACALL	code addr
32	1	RETI	

Hex Code	Number of Bytes	Mnemonic	Operands
33	1	RLC	A
34	2	ADDC	A, #data
35	2	ADDC	A, data addr
36	1	ADDC	A, @R0
37	1	ADDC	A, @R1
38	1	ADDC	A, R0
39	1	ADDC	A, R1
3A	1	ADDC	A, R2
3B	1	ADDC	A, R3
3C	1	ADDC	A, R4
3D	1	ADDC	A, R5
3E	1	ADDC	A, R6
3F	1	ADDC	A, R7
40	2	JC	code addr
41	2	AJMP	code addr
42	2	ORL	data addr, A
43	3	ORL	data addr, #data
44	2	ORL	A, #data
45	2	ORL	A, data addr
46	1	ORL	A, @R0
47	1	ORL	A, @R1
48	1	ORL	A, R0
49	1	ORL	A, R1
4A	1	ORL	A, R2
4B	1	ORL	A, R3
4C	1	ORL	A, R4
4D	1	ORL	A, R5
4E	1	ORL	A, R6
4F	1	ORL	A, R7
50	2	JNC	code addr
51	2	ACALL	code addr
52	2	ANL	data addr, A
53	3	ANL	data addr, #data
54	2	ANL	A, #data
55	2	ANL	A, data addr
56	1	ANL	A, @R0
57	1	ANL	A, @R1
58	1	ANL	A, R0
59	1	ANL	A, R1
5A	1	ANL	A, R2
5B	1	ANL	A, R3
5C	1	ANL	A, R4
5D	1	ANL	A, R5
5E	1	ANL	A, R6
5F	1	ANL	A, R7
60	2	JZ	code addr
61	2	AJMP	code addr
62	2	XRL	data addr, A
63	3	XRL	data addr, #data
64	2	XRL	A, #data
65	2	XRL	A, data addr

Table 3. Instruction Opcodes in Hexidecimal Order (Continued)

Hex Code	Number of Bytes	Mnemonic	Operands
66	1	XRL	A,@R0
67	1	XRL	A,@R1
68	1	XRL	A,R0
69	1	XRL	A,R1
6A	1	XRL	A,R2
6B	1	XRL	A,R3
6C	1	XRL	A,R4
6D	1	XRL	A,R5
6E	1	XRL	A,R6
6F	1	XRL	A,R7
70	2	JNZ	code addr
71	2	ACALL	code addr
72	2	ORL	C,bit addr
73	1	JMP	@A+DPTR
74	2	MOV	A,#data
75	3	MOV	data addr,#data
76	2	MOV	@R0,#data
77	2	MOV	@R1,#data
78	2	MOV	R0,#data
79	2	MOV	R1,#data
7A	2	MOV	R2,#data
7B	2	MOV	R3,#data
7C	2	MOV	R4,#data
7D	2	MOV	R5,#data
7E	2	MOV	R6,#data
7F	2	MOV	R7,#data
80	2	SJMP	code addr
81	2	AJMP	code addr
82	2	ANL	C,bit addr
83	1	MOVC	A,@A+PC
84	1	DIV	AB
85	3	MOV	data addr, data addr
86	2	MOV	data addr,@R0
87	2	MOV	data addr,@R1
88	2	MOV	data addr,R0
89	2	MOV	data addr,R1
8A	2	MOV	data addr,R2
8B	2	MOV	data addr,R3
8C	2	MOV	data addr,R4
8D	2	MOV	data addr,R5
8E	2	MOV	data addr,R6
8F	2	MOV	data addr,R7
90	3	MOV	DPTR,#data
91	2	ACALL	code addr
92	2	MOV	bit addr,C
93	1	MOVC	A,@A+DPTR
94	2	SUBB	A,#data
95	2	SUBB	A,data addr
96	1	SUBB	A,@R0
97	1	SUBB	A,@R1
98	1	SUBB	A,R0
99	1	SUBB	A,R1
9A	1	SUBB	A,R2
9B	1	SUBB	A,R3
9C	1	SUBB	A,R4
9D	1	SUBB	A,R5
9E	1	SUBB	A,R6
9F	1	SUBB	A,R7
A0	2	ORL	C,/bit addr
A1	2	AJMP	code addr
A2	2	MOV	C,bit addr
A3	1	INC	DPTR
A4	1	MUL	AB
A5		reserved	
A6	2	MOV	@R0,data addr
A7	2	MOV	@R1,data addr
A8	2	MOV	R0,data addr
A9	2	MOV	R1,data addr
AA	2	MOV	R2,data addr
AB	2	MOV	R3,data addr
AC	2	MOV	R4,data addr
AD	2	MOV	R5,data addr
AE	2	MOV	R6,data addr
AF	2	MOV	R7,data addr
B0	2	ANL	C,/bit addr
B1	2	ACALL	code addr
B2	2	CPL	bit addr
B3	1	CPL	C
B4	3	CJNE	A,#data,code addr
B5	3	CJNE	A,data addr,code addr
B6	3	CJNE	@R0,#data,code addr
B7	3	CJNE	@R1,#data,code addr
B8	3	CJNE	R0,#data,code addr
B9	3	CJNE	R1,#data,code addr
BA	3	CJNE	R2,#data,code addr
BB	3	CJNE	R3,#data,code addr
BC	3	CJNE	R4,#data,code addr
BD	3	CJNE	R5,#data,code addr
BE	3	CJNE	R6,#data,code addr
BF	3	CJNE	R7,#data,code addr
C0	2	PUSH	data addr
C1	2	AJMP	code addr
C2	2	CLR	bit addr
C3	1	CLR	C
C4	1	SWAP	A
C5	2	XCH	A,data addr
C6	1	XCH	A,@R0
C7	1	XCH	A,@R1
C8	1	XCH	A,R0
C9	1	XCH	A,R1
CA	1	XCH	A,R2
CB	1	XCH	A,R3



Table 3. Instruction Opcodes in Hexidecimal Order (Continued)

Hex Code	Number of Bytes	Mnemonic	Operands
CC	1	XCH	A,R4
CD	1	XCH	A,R5
CE	1	XCH	A,R6
CF	1	XCH	A,R7
D0	2	POP	data addr
D1	2	ACALL	code addr
D2	2	SETB	bit addr
D3	1	SETB	C
D4	1	DA	A
D5	3	DJNZ	data addr,code addr
D6	1	XCHD	A,@R0
D7	1	XCHD	A,@R1
D8	2	DJNZ	R0,code addr
D9	2	DJNZ	R1,code addr
DA	2	DJNZ	R2,code addr
DB	2	DJNZ	R3,code addr
DC	2	DJNZ	R4,code addr
DD	2	DJNZ	R5,code addr
DE	2	DJNZ	R6,code addr
DF	2	DJNZ	R7,code addr
E0	1	MOVX	A,@DPTR
E1	2	AJMP	code addr
E2	1	MOVX	A,@R0
E3	1	MOVX	A,@R1
E4	1	CLR	A
E5	2	MOV	A,data addr
E6	1	MOV	A,@R0
E7	1	MOV	A,@R1
E8	1	MOV	A,R0
E9	1	MOV	A,R1
EA	1	MOV	A,R2
EB	1	MOV	A,R3
EC	1	MOV	A,R4
ED	1	MOV	A,R5
EE	1	MOV	A,R6
EF	1	MOV	A,R7
F0	1	MOVX	@DPTR,A
F1	2	ACALL	code addr
F2	1	MOVX	@R0,A
F3	1	MOVX	@R1,A
F4	1	CPL	A
F5	2	MOV	data addr,A
F6	1	MOV	@R0,A
F7	1	MOV	@R1,A
F8	1	MOV	R0,A
F9	1	MOV	R1,A
FA	1	MOV	R2,A
FB	1	MOV	R3,A
FC	1	MOV	R4,A
FD	1	MOV	R5,A
FE	1	MOV	R6,A
FF	1	MOV	R7,A

# 8031AH/8051AH 8032AH/8052AH 8751H/8751H

## EXPRESS

- Extended Temperature Range
- Burn-in

The Intel EXPRESS system offers enhancements to the operational specifications of the MCS-51 family of microcontrollers. These EXPRESS products are designed to meet the needs of those applications whose operating requirements exceed commercial standards.

The EXPRESS program includes the commercial standard temperature range with burn-in, and an extended temperature range with or without burn-in.

With the commercial standard temperature range operational characteristics are guaranteed over the temperature range of 0°C to 70°C. With the extended temperature range option, operational characteristics are guaranteed over the range of -40°C to +85°C.

The optional burn-in is dynamic, for a minimum time of 160 hours at 125°C with  $V_{CC} = 5.5V \pm 0.5V$ , following guidelines in MIL-STD-883, Method 1015.

Package types and EXPRESS versions are identified by a one- or two-letter prefix to the part number. The prefixes are listed in Table 1.

For the extended temperature range option, this data sheet specifies the parameters which deviate from their commercial temperature range limits. The commercial temperature range data sheets are applicable for all parameters not listed here.

## Electrical Deviations from Commercial Specifications for Extended Temperature Range

D.C. and A.C. parameters not included here are the same as in the commercial temperature range data sheets.

### D.C. CHARACTERISTICS: ( $T_A = -40^\circ\text{C}$ to $+85^\circ\text{C}$ ; $V_{CC} = 5V \pm 10\%$ ; $V_{SS} = 0V$ )

Symbol	Parameter	Min	Max	Unit	Test Conditions
$V_{IL}$	Input Low Voltage	-0.5	0.75	V	
$V_{IH}$	Input High Voltage (Except XTAL2, RST)	2.1	$V_{CC} + 0.5$	V	
$I_{CC}$	Power Supply Current: 8051AH,8031AH 8052AH,8032AH 8751H,8751H		135 175 265	ma ma ma	All Outputs Disconnected; $\bar{E}A = V_{CC}$

Table 1 — Prefix Identification

Prefix	Package Type	Temperature Range	Burn-In
P	plastic	commercial	no
D	cerdip	commercial	no
C	ceramic	commercial	no
TP	plastic	extended	no
TD	cerdip	extended	no
TC	ceramic	extended	no
QP	plastic	commercial	yes
QD	cerdip	commercial	yes
QC	ceramic	commercial	yes
LP	plastic	extended	yes
LD	cerdip	extended	yes
LC	ceramic	extended	yes

Please note:

- Commercial temperature range is 0° to 70°C. Extended temperature range is -40° to +85°C.
- Burn-in is dynamic, for a minimum time of 160 hours at 125°C,  $V_{CC} = 5.5V \pm 0.5V$ , following guidelines in MIL-STD-883 Method 1015 (Test Condition D).
- The following devices are not available in plastic packages:  
8751H, 8751H
- The following devices are not available in ceramic packages:  
8051AH, 8031AH  
8052AH, 8032AH

Examples: P8031AH indicates 8031AH in a plastic package and specified for commercial temperature range, without burn-in. LD8751H indicates 8751H in a cerdip package and specified for extended temperature range with burn-in.

# Electrical Deviations from Commercial Specifications for Extended Temperature Range

D.C. and A.C. parameters not included here are the same as in the commercial temperature range data sheets.

D.C. CHARACTERISTICS:  $T_A = -40^\circ\text{C}$  to  $+85^\circ\text{C}$ ;  $V_{CC} = 5V \pm 10\%$ ;  $V_{SS} = 0V$

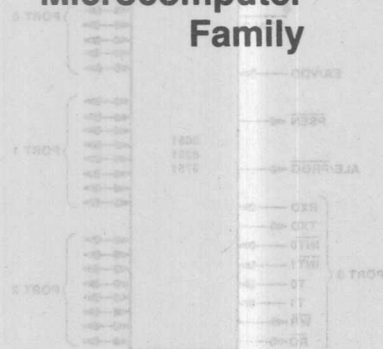
Symbol	Parameter	Min	Max	Unit	Test Conditions
$V_L$	Input Low Voltage	-0.5	-0.75	V	
$V_{IH}$	Input High Voltage (Except XTALS, RST)	2.1	$V_{CC} + 0.5$	V	
$I_{CC}$	Power Supply Current 8051AH, 8031AH 8052AH, 8032AH 8751H, 8751H		135 175 285	mA mA mA	All Outputs Disconnected; $E_A = V_{CC}$







# An Introduction to the Intel MCS®-51 Single-Chip Microcomputer Family



## Contents

<b>1. INTRODUCTION</b>	10-2
Family Overview	10-2
Microcomputer Background Concepts	10-3
<b>2. ARCHITECTURE AND ORGANIZATION</b>	10-5
Central Processing Unit	10-6
Memory Spaces	10-9
Input/Output Ports	10-10
Special Peripheral Functions	10-11
<b>3. INSTRUCTION SET AND ADDRESSING MODES</b>	10-15
Data Addressing Modes	10-15
Addressing Mode Combinations	10-18
Advantages of Symbolic Addressing	10-18
Arithmetic Instruction Usage	10-19
Multiplication and Division	10-20
Logical Byte Operations	10-20
Program Control	10-21
Operate-and-Branch Instructions	10-22
Stack Operations	10-22
Table Look-Up Instructions	10-23
<b>4. BOOLEAN PROCESSING INSTRUCTIONS</b>	10-25
Direct Bit Addressing	10-25
Bit Manipulation Instructions	10-25
Solving Combinatorial Logic Equations	10-26
<b>5. ON-CHIP PERIPHERAL FUNCTIONS</b>	10-28
I/O Ports	10-28
Serial Port and Timer	10-29
<b>6. SUMMARY</b>	10-30

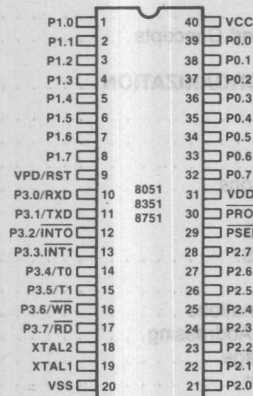


Figure 1a. 8051 Microcomputer Pinout Diagram

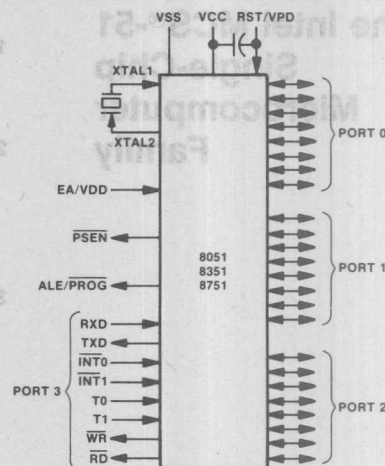


Figure 1b. 8051 Microcomputer Logic Symbol

## 1. INTRODUCTION

In 1976 Intel introduced the MCS-48™ family, consisting of the 8048, 8748, and 8035 microcomputers. These parts marked the first time a complete microcomputer system, including an eight-bit CPU, 1024 8-bit words of ROM or EPROM program memory, 64 words of data memory, I/O ports and an eight-bit timer/counter could be integrated onto a single silicon chip. Depending only on the program memory contents, one chip could control a limitless variety of products, ranging from appliances or automobile engines to text or data processing equipment. Follow-on products stretched the MCS-48™ architecture in several directions: the 8049 and 8039 doubled the amount of on-chip memory and ran 83% faster; the 8021 reduced costs by executing a subset of the 8048 instructions with a somewhat slower clock; and the 8022 put a unique two-channel 8-bit analog-to-digital converter on the same NMOS chip as the computer, letting the chip interface directly with analog transducers.

Now three new high-performance single-chip microcomputers—the Intel® 8051, 8751, and 8031—extend the advantages of Integrated Electronics to whole new product areas. Thanks to Intel's new HMOS technology, the MCS-51™ family provides four times the program memory and twice the data memory as the 8048 on a single chip. New I/O and peripheral capabilities both increase the range of applicability and reduce total system cost. Depending on the use, processing throughput increases by two and one-half to ten times.

This Application Note is intended to introduce the reader to the MCS-51™ architecture and features. While it does not assume intimacy with the MCS-48™ product line on the part of the reader, he/she should be familiar with

some microprocessor (preferably Intel's, of course) or have a background in computer programming and digital logic.

## Family Overview

Pinout diagrams for the 8051, 8751, and 8031 are shown in Figure 1. The devices include the following features:

- Single-supply 5 volt operation using HMOS technology.
- 4096 bytes program memory on-chip (not on 8031).
- 128 bytes data memory on-chip.
- Four register banks.
- 128 User-defined software flags.
- 64 Kilobytes each program and external RAM addressability.
- One microsecond instruction cycle with 12 MHz crystal.
- 32 bidirectional I/O lines organized as four 8-bit ports (16 lines on 8031).
- Multiple mode, high-speed programmable Serial Port.
- Two multiple mode, 16-bit Timer/Counters.
- Two-level prioritized interrupt structure.
- Full depth stack for subroutine return linkage and data storage.
- Augmented MCS-48™ instruction set.
- Direct Byte and Bit addressability.
- Binary or Decimal arithmetic.
- Signed-overflow detection and parity computation.
- Hardware Multiple and Divide in 4  $\mu$ sec.
- Integrated Boolean Processor for control applications.
- Upwardly compatible with existing 8048 software.

All three devices come in a standard 40-pin Dual In-Line Package, with the same pin-out, the same timing, and the same electrical characteristics. The primary difference between the three is the on-chip program memory—different types are offered to satisfy differing user requirements.

The 8751 provides 4K bytes of ultraviolet-Erasable, Programmable Read Only Memory (EPROM) for program development, prototyping, and limited production runs. (By convention, 1K means  $2^{10} = 1024$ . 1k—with a lower case “k”—equals  $10^3 = 1000$ .) This part may be individually programmed for a specific application using Intel's Universal PROM Programmer (UPP). If software bugs are detected or design specifications change the same part may be “erased” in a matter of minutes by exposure to ultraviolet light and reprogrammed with the modified code. This cycle may be repeated indefinitely during the design and development phase.

The final version of the software must be programmed into a large number of production parts. The 8051 has 4K bytes of ROM which are mask-programmed with the customer's order when the chip is built. This part is considerably less expensive, but cannot be erased or altered after fabrication.

The 8031 does not have any program memory on-chip, but may be used with up to 64K bytes of external standard or multiplexed ROMs, PROMs, or EPROMs. The 8031 fits well in applications requiring significantly larger or smaller amounts of memory than the 4K bytes provided by its two siblings.

(The 8051 and 8751 automatically access external program memory for all addresses greater than the 4096 bytes on-chip. The External Access input is an override for all internal program memory—the 8051 and 8751 will each emulate an 8031 when pin 31 is low.)

Throughout this Note, “8051” is used as a generic term. Unless specifically stated otherwise, the point applies equally to all three components. Table 1 summarizes the quantitative differences between the members of the MCS-48™ and MCS-51™ families.

The remainder of this Note discusses the various MCS-51™ features and how they can be used. Software and/or hard-

ware application examples illustrate many of the concepts. Several isolated tasks (rather than one complete system design example) are presented in the hope that some of them will apply to the reader's experiences or needs.

A document this short cannot detail all of a computer system's capabilities. By no means will all the 8051 instructions be demonstrated; the intent is to stress new or unique MCS-51™ operations and instructions generally used in conjunction with each other. For additional hardware information refer to the Intel MCS-51™ Family User's Manual, publication number 121517. The assembly language and use of ASM51, the MCS-51™ assembler, are further described in the MCS-51™ Macro Assembler User's Guide, publication number 9800937.

The next section reviews some of the basic concepts of microcomputer design and use. Readers familiar with the 8048 may wish to skim through this section or skip directly to the next, “ARCHITECTURE AND ORGANIZATION.”

## Microcomputer Background Concepts

Most digital computers use the binary (base 2) number system internally. All variables, constants, alphanumeric characters, program statements, etc., are represented by groups of binary digits (“bits”), each of which has the value 0 or 1. Computers are classified by how many bits they can move or process at a time.

The MCS-51™ microcomputers contain an eight-bit central processing unit (CPU). Most operations process variables eight bits wide. All internal RAM and ROM, and virtually all other registers are also eight bits wide. An eight-bit (“byte”) variable (shown in Figure 2) may assume one of  $2^8 = 256$  distinct values, which usually represent integers between 0 and 255. Other types of numbers, instructions, and so forth are represented by one or more bytes using certain conventions.

For example, to represent positive and negative values, the most significant bit (D7) indicates the sign of the other seven bits—0 if positive, 1 if negative—allowing integer variables, between -128 and +127. For integers with extremely large magnitudes, several bytes are manipulated together as “multiple precision” signed or unsigned integers—16, 24, or more bits wide.

Table 1. Features of Intel's Single-Chip Microcomputers

EPROM Program Memory	ROM Program Memory	External Program Memory	Program Memory (Int/Max)	Data Memory (Bytes)	Instr. Cycle Time	Input/Output Pins	Interrupt Sources	Reg. Banks
—	8021	—	1K/1K	64	8.4 $\mu$ Sec	21	0	1
—	8022	—	2K/2K	64	8.4 $\mu$ Sec	28	2	1
8748	8048	8035	1K/4K	64	2.5 $\mu$ Sec	27	2	2
—	8049	8039	2K/4K	128	1.36 $\mu$ Sec	27	2	2
8751	8051	8031	4K/64K	128	1.0 $\mu$ Sec	32	5	4

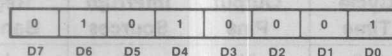


The letters "MCS" have traditionally indicated a system or family of compatible Intel® microcomputer components, including CPUs, memories, clock generators, I/O expanders, and so forth. The numerical suffix indicates the microprocessor or microcomputer which serves as the cornerstone of the family. Microcomputers in the MCS-48™ family currently include the 8048-series (8035, 8048, & 8748), the 8049-series (8039 & 8049), and the 8021 and 8022; the family also includes the 8243, an I/O expander compatible with each of the microcomputers. Each computer's CPU is derived from the 8048, with essentially the same architecture, addressing modes, and instruction set, and a single assembler (ASM48) serves each.

The first members of the MCS-51™ family are the 8051, 8751, and 8031. The architecture of the 8051-series, while derived from the 8048, is not strictly compatible; there are more addressing modes, more instructions, larger address spaces, and a few other hardware differences. In this Application Note the letters "MCS-51" are used when referring to architectural features of the 8051-series—features which would be included on possible future microcomputers based on the 8051 CPU. Such products could have different amounts of memory (as in the 8048/8049) or different peripheral functions (as in the 8021 and 8022) while leaving the CPU and instruction set intact. ASM51 is the assembler used by all microcomputers in the 8051 family.

Two digit decimal numbers may be "packed" in an eight-bit value, using four bits for the binary code of each digit. This is called Binary-Coded Decimal (BCD) representation, and is often used internally in programs which interact heavily with human beings.

Alphanumeric characters (letters, numbers, punctuation marks, etc.) are often represented using the American Standard Code for Information Interchange (ASCII) convention. Each character is associated with a unique seven-bit binary number. Thus one byte may represent



**Figure 2. Representation of Bits Within an Eight-Bit "Byte" (Value shown = 01010001 Binary = 81 decimal).**

a single character, and a word or sequence of letters may be represented by a series (or "string") of bytes. Since the ASCII code only uses 128 characters, the most significant bit of the byte is not needed to distinguish between characters. Often D7 is set to 0 for all characters. In some coding schemes, D7 is used to indicate the "parity" of the other seven bits—set or cleared as necessary to ensure that the total number of "1" bits in the eight-bit code is even ("even parity") or odd ("odd parity"). The 8051 includes hardware to compute parity when it is needed.

A computer program consists of an ordered sequence of specific, simple steps to be executed by the CPU one-at-a-time. The method or sequence of steps used collectively to solve the user's application is called an "algorithm."

The program is stored inside the computer as a sequence of binary numbers, where each number corresponds to one of the basic operations ("opcodes") which the CPU is capable of executing. In the 8051, each program memory location is one byte. A complete instruction consists of a sequence of one or more bytes, where the first defines the operation to be executed and additional bytes (if needed) hold additional information, such as data values or variable addresses. No instruction is longer than three bytes.

The way in which binary opcodes and modifier bytes are assigned to the CPU's operations is called the computer's "machine language." Writing a program directly in machine language is time-consuming and tedious. Human beings think in words and concepts rather than encoded numbers, so each CPU operation and resource is given a name and standard abbreviation ("mnemonic"). Programs are more easily discussed using these standard mnemonics, or "assembly language," and may be typed into an Intel® Intellec® 800 or Series II® microcomputer development system in this form. The development system can mechanically translate the program from assembly language "source" form to machine language "object" code using a program called an "assembler." The MCS-51™ assembler is called ASM51.

There are several important differences between a computer's machine language and the assembly language used as a tool to represent it. The machine language or instruction set is the set of operations which the CPU can perform while a program is executing ("at run-time"), and is strictly determined by the microcomputer hardware design.

The assembly language is a standard (though more-or-less arbitrary) set of symbols including the instruction set mnemonics, but with additional features which further simplify the program design process. For example, ASM51 has controls for creating and formatting a program listing, and a number of directives for allocating variable storage and inserting arbitrary bytes of data into the object code for creating tables of constants.

In addition, ASM51 can perform sophisticated mathematical operations, computing addresses or evaluating arithmetic expressions to relieve the programmer from this drudgery. However, these calculations can only use information known at "assembly time."

For example, the 8051 performs arithmetic calculations at run-time, eight bits at a time. ASM51 can do similar operations 16 bits at a time. The 8051 can only do one simple step per instruction, while ASM51 can perform complex calculations in each line of source code. However, the operations performed by the assembler may only use parameter values fixed at assembly-time, not variables whose values are unknown until program execution begins.

For example, when the assembly language source line,

```
ADD A,#(LOOP_COUNT + 1) * 3
```

is assembled, ASM51 will find the value of the previously-defined constant "LOOP\_COUNT" in an internal symbol table, increment the value, multiply the sum by three, and (assuming it is between -256 and 255 inclusive) truncate the product to eight bits. When this instruction is executed, the 8051 ALU will just add that resulting constant to the accumulator.

Some similar differences exist to distinguish number system ("radix") specifications. The 8051 does all computations in binary (though there are provisions for then converting the result to decimal form). In the course of writing a program, though, it may be more convenient to specify constants using some other radix, such as base 10. On other occasions, it is desirable to specify the ASCII code for some character or string of characters without referring to tables. ASM51 allows several representations for constants, which are converted to binary as each instruction is assembled.

For example, binary numbers are represented in the

assembly language by a series of ones and zeros (naturally), followed by the letter "B" (for Binary); octal numbers as a series of octal digits (0-7) followed by the letter "O" (for Octal) or "Q" (which doesn't stand for anything, but looks sort of like an "O" and is less likely to be confused with a zero).

Hexadecimal numbers are represented by a series of hexadecimal digits (0-9,A-F), followed by (you guessed it) the letter "H." A "hex" number must begin with a decimal digit; otherwise it would look like a user-defined symbol (to be discussed later). A "dummy" leading zero may be inserted before the first digit to meet this constraint. The character string "BACH" could be a legal label for a Baroque music synthesis routine; the string "0BACH" is the hexadecimal constant BACH<sub>16</sub>. This is a case where adding 0 makes a big difference.

Decimal numbers are represented by a sequence of decimal digits, optionally followed by a "D." If a number has no suffix, it is assumed to be decimal—so it had better not contain any non-decimal digits. "0BAC" is not a legal representation for anything.

When an ASCII code is needed in a program, enclose the desired character between two apostrophes (as in '#') and the assembler will convert it to the appropriate code (in this case 23H). A string of characters between apostrophes is translated into a series of constants; 'BACH' becomes 42H, 41H, 43H, 48H.

These same conventions are used throughout the associated Intel documentation. Table 2 illustrates some of the different number formats.

## 2. ARCHITECTURE AND ORGANIZATION

Figure 3 blocks out the MCS-51™ internal organization. Each microcomputer combines a Central Processing Unit, two kinds of memory (data RAM plus program ROM or EPROM), Input/Output ports, and the mode,

Table 2. Notations Used to Represent Numbers

Bit Pattern	Binary	Octal	Hexa-Decimal	Decimal	Signed Decimal
0 0 0 0 0 0 0 0	0B	0Q	00H	0	0
0 0 0 0 0 0 0 1	1B	1Q	01H	1	+1
...	...	...	...	...	...
0 0 0 0 0 1 1 1	111B	7Q	07H	7	+7
0 0 0 0 1 0 0 0	1000B	10Q	08H	8	+8
0 0 0 0 1 0 0 1	1001B	11Q	09H	9	+9
0 0 0 0 1 0 1 0	1010B	12Q	0AH	10	+10
...	...	...	...	...	...
0 0 0 0 1 1 1 1	1111B	17Q	0FH	15	+15
0 0 0 1 0 0 0 0	10000B	20Q	10H	16	+16
...	...	...	...	...	...
0 1 1 1 1 1 1 1	111111B	177Q	7FH	127	+127
1 0 0 0 0 0 0 0	1000000B	200Q	80H	128	-128
1 0 0 0 0 0 0 1	1000001B	201Q	81H	129	-127
...	...	...	...	...	...
1 1 1 1 1 1 1 0	1111110B	376Q	0FEH	254	-2
1 1 1 1 1 1 1 1	1111111B	377Q	0FFH	255	-1

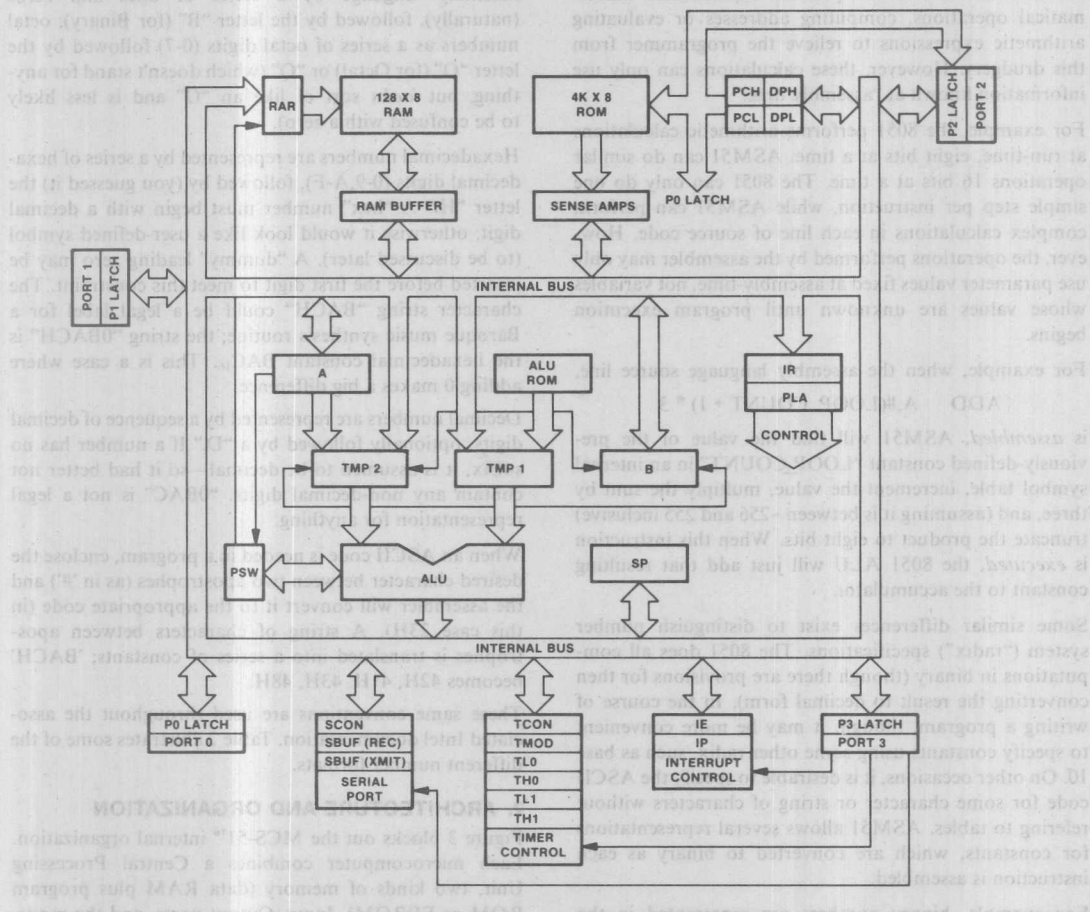


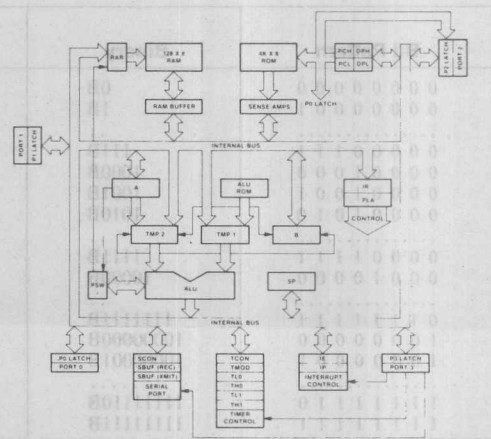
Figure 3. Block Diagram of 8051 Internal Structure

status, and data registers and random logic needed for a variety of peripheral functions. These elements communicate through an eight-bit data bus which runs throughout the chip, somewhat akin to indoor plumbing. This bus is buffered to the outside world through an I/O port when memory or I/O expansion is desired.

Let's summarize what each block does; later chapters dig into the CPU's instruction set and the peripheral registers in much greater detail.

### Central Processing Unit

The CPU is the "brains" of the microcomputer, reading the user's program and executing the instructions stored therein. Its primary elements are an eight-bit Arithmetic/Logic Unit with associated registers A, B, PSW, and SP, and the sixteen-bit Program Counter and "Data Pointer" registers.





### Arithmetic Logic Unit

The ALU can perform (as the name implies) arithmetic and logic functions on eight-bit variables. The former include basic addition, subtraction, multiplication, and division; the latter include the logical operations AND, OR, and Exclusive-OR, as well as rotate, clear, complement, and so forth. The ALU also makes conditional branching decisions, and provides data paths and temporary registers used for data transfers within the system. Other instructions are built up from these primitive functions: the addition capability can increment registers or automatically compute program destination addresses; subtraction is also used in decrementing or comparing the magnitude of two variables.

These primitive operations are automatically cascaded and combined with dedicated logic to build complex instructions such as incrementing a sixteen-bit register pair. To execute one form of the compare instruction, for example, the 8051 increments the program counter three times, reads three bytes of program memory, computes a register address with logical operations, reads internal data memory twice, makes an arithmetic comparison of two variables, computes a sixteen-bit destination address, and decides whether or not to make a branch—all in two microseconds!

An important and unique feature of the MCS-51 architecture is that the ALU can also manipulate one-bit as well as eight-bit data types. Individual bits may be set, cleared, or complemented, moved, tested, and used in logic computations. While support for a more primitive data type may initially seem a step backwards in an era of increasing word length, it makes the 8051 especially well suited for controller-type applications. Such algorithms *inherently* involve Boolean (true/false) input and output variables, which were heretofore difficult to implement with standard microprocessors. These features are collectively referred to as the MCS-51™ “Boolean Processor,” and are described in the so-named chapter to come.

Thanks to this powerful ALU, the 8051 instruction set fares well at both real-time control and data intensive algorithms. A total of 51 separate operations move and manipulate three data types: Boolean (1-bit), byte (8-bit), and address (16-bit). All told, there are eleven addressing modes—seven for data, four for program sequence control (though only eight are used by more than just a few specialized instructions). Most operations allow several addressing modes, bringing the total number of instructions (operation/addressing mode combinations) to 111, encompassing 255 of the 256 possible eight-bit instruction opcodes.

### Instruction Set Overview

Table 4 lists these 111 instructions classified into five groups:

- Arithmetic Operations
- Logical Operations for Byte Variables
- Data Transfer Instructions
- Boolean Variable Manipulation
- Program Branching and Machine Control

MCS-48™ programmers perusing Table 4 will notice the absence of special categories for Input/Output, Timer/Counter, or Control instructions. These functions are all still provided (and indeed many new functions are added), but as special cases of more generalized operations in other categories. To explicitly list all the useful instructions involving I/O and peripheral registers would require a table approximately four times as long.

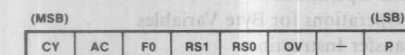
Observant readers will also notice that all of the 8048's page-oriented instructions (conditional jumps, JMPP, MOVP, MOVP3) have been replaced with corresponding but non-paged instructions. The 8051 instruction set is entirely *non-page-oriented*. The MCS-48™ “MOVP” instruction replacement and all conditional jump instructions operate relative to the program counter, with the actual jump address computed by the CPU during instruction execution. The “MOVP3” and “JMPP” replacements are now made relative to another sixteen-bit register, which allows the effective destination to be anywhere in the program memory space, regardless of where the instruction itself is located. There are even three-byte jump and call instructions allowing the destination to be *anywhere* in the 64K program address space.

The instruction set is designed to make programs efficient both in terms of code size and execution speed. No instruction requires more than three bytes of program memory, with the majority requiring only one or two bytes. Virtually all instructions execute in either one or two instruction cycles—one or two microseconds with a 12-MHz crystal—with the sole exceptions (multiply and divide) completing in four cycles.

Many instructions such as arithmetic and logical functions or program control, provide both a short and a long form for the same operation, allowing the programmer to optimize the code produced for a specific application. The 8051 usually fetches two instruction bytes per instruction cycle, so using a shorter form can lead to faster execution as well.

For example, any byte of RAM may be loaded with a constant with a three-byte, two-cycle instruction, but the commonly used “working registers” in RAM may be initialized in one cycle with a two-byte form. Any bit anywhere on the chip may be set, cleared, or complemented by a single three-byte logical instruction using two cycles. But critical control bits, I/O pins, and software flags may be controlled by two-byte, single cycle instructions. While three-byte jumps and calls can “go anywhere” in program memory, nearby sections of code may be reached by shorter relative or absolute versions.





# Symbol Position Name and Significance

CY	PSW.7	Carry flag. Set/cleared by hardware or software during certain arithmetic and logical instructions.
AC	PSW.6	Auxiliary Carry flag. Set/cleared by hardware during addition or subtraction instructions to indicate carry or borrow out of bit 3.
F0	PSW.5	Flag 0 Set/cleared/tested by software as a user-defined status flag.
RS1	PSW.4	Register bank Select control bits 1 & 0. Set/cleared by software to determine working register bank (see Note).
RS	PSW.3	

# Symbol Position Name and Significance

OV PSW.2 Overflow flag.  
Set/cleared by hardware during arithmetic instructions to indicate overflow conditions.

PSW.1 (reserved)

P PSW.0 Parity flag.  
Set/cleared by hardware each instruction cycle to indicate an odd/even number of "one" bits in the accumulator, i.e., even parity.

**Note—** the contents of (RS1, RS0) enable the working register banks as follows:

(0,0)—Bank 0	(00H-07H)
(0,1)—Bank 1	(08H-0FH)
(1,0)—Bank 2	(10H-17H)
(1,1)—Bank 3	(18H-1FH)

**Figure 4. PSW—Program Status Word Organization**

A significant side benefit of an instruction set more powerful than those of previous single-chip microcomputers is that it is easier to generate applications-oriented software. Generalized addressing modes for byte and bit instructions reduce the number of source code lines written and debugged for a given application. This leads in turn to proportionately lower software costs, greater reliability, and faster design cycles.

## Accumulator and PSW

The 8051, like its 8048 predecessor, is primarily an accumulator-based architecture: an eight-bit register called the accumulator ("A") holds a source operand and receives the result of the arithmetic instructions (addition, subtraction, multiplication, and division). The accumulator can be the source or destination for logical operations and a number of special data movement instructions, including table look-ups and external RAM expansion. Several functions apply exclusively to the accumulator: rotates, parity computation, testing for zero, and so on.

Many instructions implicitly or explicitly affect (or are affected by) several status flags, which are grouped together to form the Program Status Word shown in Figure 4.

(The period within entries under the Position column is called the "dot operator," and indicates a particular bit position within an eight-bit byte. "PSW.5" specifies bit 5 of the PSW. Both the documentation and ASM51 use this notation.)

The most "active" status bit is called the carry flag (abbreviated "CY"). This bit makes possible multiple precision arithmetic operations including addition, subtraction,

and rotates. The carry also serves as a "Boolean accumulator" for one-bit logical operations and bit manipulation instructions. The overflow flag (OV) detects when arithmetic overflow occurs on signed integer operands, making two's complement arithmetic possible. The parity flag (P) is updated after every instruction cycle with the even-parity of the accumulator contents.

The CPU does not control the two register-bank select bits, RS1 and RS0. Rather, they are manipulated by software to enable one of the four register banks. The usage of the PSW flags is demonstrated in the Instruction Set chapter of this Note.

Even though the architecture is accumulator-based, provisions have been made to bypass the accumulator in common instruction situations. Data may be moved from any location on-chip to any register, address, or indirect address (and vice versa), any register may be loaded with a constant, etc., all without affecting the accumulator. Logical operations may be performed against registers or variables to alter fields of bits—without using or affecting the accumulator. Variables may be incremented, decremented, or tested without using the accumulator. Flags and control bits may be manipulated and tested without affecting anything else.

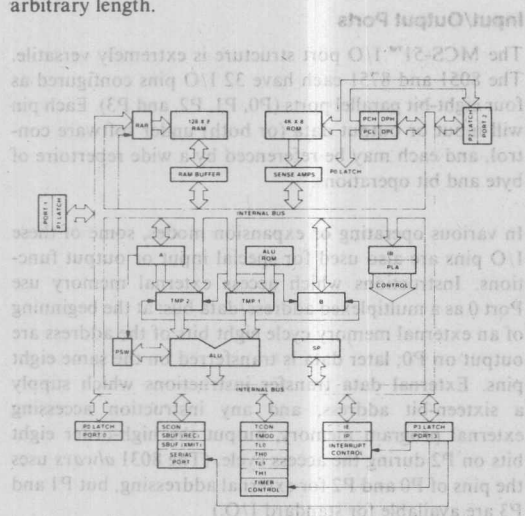
## Other CPU Registers

A special eight-bit register ("B") serves in the execution of the multiply and divide instructions. This register is used in conjunction with the accumulator as the second input operand and to return eight-bits of the result.

The MCS-51 family processors include a hardware stack within internal RAM, useful for subroutine linkage.

passing parameters between routines, temporary variable storage, or saving status during interrupt service routines. The Stack Pointer (SP) is an eight-bit pointer register which indicates the address of the last byte pushed onto the stack. The stack pointer is automatically incremented or decremented on all push or pop instructions and all subroutine calls and returns. In theory, the stack in the 8051 may be up to a full 128 bytes deep. (In practice, even simple programs would use a handful of RAM locations for pointers, variables, and so forth—reducing the stack depth by that number.) The stack pointer defaults to 7 on reset, so that the stack will start growing up from location 8, just like in the 8048. By altering the pointer contents the stack may be relocated anywhere within internal RAM.

Finally, a 16-bit register called the data pointer (DPTR) serves as a base register in indirect jumps, table look-up instructions, and external data transfers. The high- and low-order halves of the data pointer may be manipulated as separate registers (DPH and DPL, respectively) or together using special instructions to load or increment all sixteen bits. Unlike the 8048, look-up tables can therefore start anywhere in program memory and be of arbitrary length.



## Memory Spaces

Program memory is separate and distinct from data memory. Each memory type has a different addressing mechanism, different control signals, and a different function.

The program memory array (ROM or EPROM), like an elephant, is extremely large and never forgets information, even when power is removed. Program memory is used for information needed each time power is applied: initialization values, calibration constants, keyboard layout tables, etc., as well as the program itself. The program memory has a sixteen-bit address bus; its elements

are addressed using the Program Counter or instructions which generate a sixteen-bit address.

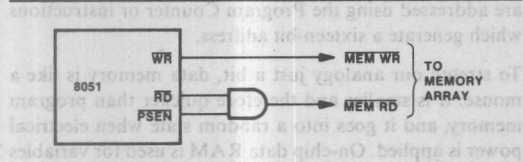
To stretch our analogy just a bit, data memory is like a mouse: it is smaller and therefore quicker than program memory, and it goes into a random state when electrical power is applied. On-chip data RAM is used for variables which are determined or may change while the program is running.

A computer spends most of its time manipulating variables, not constants, and a relatively small number of variables at that. Since eight-bits is more than sufficient to uniquely address 128 RAM locations, the on-chip RAM address register is only one byte wide. In contrast to the program memory, data memory accesses need a single eight-bit value—a constant or another variable—to specify a unique location. Since this is the basic width of the ALU and the different memory types, those resources can be used by the addressing mechanisms, contributing greatly to the computer's operating efficiency.

The partitioning of program and data memory is extended to off-chip memory expansion. Each may be added independently, and each uses the same address and data busses, but with different control signals. External program memory is gated onto the external data bus by the PSEN (Program Store Enable) control output, pin 29. External data memory is read onto the bus by the  $\overline{RD}$  output, pin 17, and written with data supplied from the microcomputer by the  $\overline{WR}$  output, pin 16. (There is no control pin to write external program ROM, which is by definition Read Only.) While both types may be expanded to up to 64K bytes, the external data memory may optionally be expanded in 256 byte "pages" to preserve the use of P2 as an I/O port. This is useful with a relatively small expansion RAM (such as the Intel® 8155) or for addressing external peripherals.

Single-chip controller programs are finalized during the project design cycle, and are not modified after production. Intel's single-chip microcomputers are not "von Neumann" architectures common among main-frame and mini-computer systems: the MCS-51™ processor data memory—on-chip and external—may not be used for program code. Just as there is no write-control signal for program memory, there is no way for the CPU to execute instructions out of RAM. In return, this concession allows an architecture optimized for efficient controller applications: a large, fixed program located in ROM, a hundred or so variables in RAM, and different methods for efficiently addressing each.

(Von Neumann machines are helpful for software development and debug. An 8051 system could be modified to have a single off-chip memory space by gating together the two memory-read controls ( $\overline{PSEN}$  and  $\overline{RD}$ ) with a two-input AND gate (Figure 5). The CPU could then write data into the common memory array using  $\overline{WR}$  and



### Figure 5. Combining External Program and Data Memory Arrays

external data transfer instructions, and read instructions or data with the AND gate output and data transfer or program memory look-up instructions.)

In addition to the memory arrays, there is (yet) another (albeit sparsely populated) physical address space. Connected to the internal data bus are a score of special-purpose eight-bit registers scattered throughout the chip. Some of these—B, SP, PSW, DPH, and DPL—have been discussed above. Others—I/O ports and peripheral function registers—will be introduced in the following sections. Collectively, these registers are designated as the “special-function register” address space. Even the accumulator is assigned a spot in the special-function register address space for additional flexibility and uniformity.

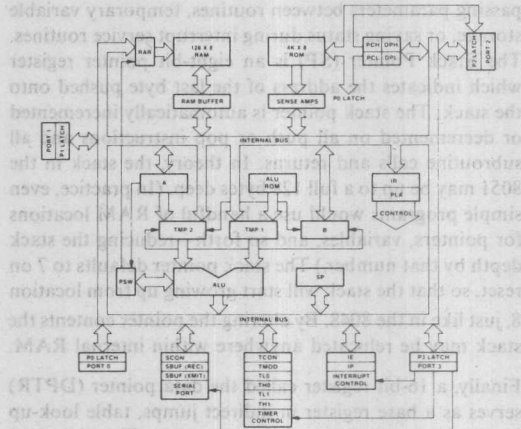
Thus, the MCS-51™ architecture supports several distinct “physical” address spaces, functionally separated at the hardware level by different addressing mechanisms, read and write control signals, or both:

- On-chip program memory;
- On-chip data memory;
- Off-chip program memory;
- Off-chip data memory;
- On-chip special-function registers.

What the *programmer sees*, though, are "logical" address spaces. For example, as far as the programmer is concerned, there is only one type of program memory, 64K bytes in length. The fact that it is formed by combining on- and off-chip arrays (split 4K/60K on the 8051 and 8751) is "invisible" to the programmer; the CPU automatically fetches each byte from the appropriate array, based on its address.

(Presumably, future microcomputers based on the MCS-51™ architecture may have a different physical split, with more or less of the 64K total implemented on-chip. Using the MCS-48™ family as a precedent, the 8048's 4K potential program address space was split 1K/3K between on- and off-chip arrays; the 8049's was split 2K/2K.)

Why go into such tedious details about address spaces? The logical addressing modes are described in the Instruction Set chapter in terms of physical address spaces. Understanding their differences now will pay off in understanding and using the chips later.



### Input/Output Ports

The MCS-51™ I/O port structure is extremely versatile. The 8051 and 8751 each have 32 I/O pins configured as four eight-bit parallel ports (P0, P1, P2, and P3). Each pin will input or output data (or both) under software control, and each may be referenced by a wide repertoire of byte and bit operations.

In various operating or expansion modes, some of these I/O pins are also used for special input or output functions. Instructions which access external memory use Port 0 as a multiplexed address/data bus: at the beginning of an external memory cycle eight bits of the address are output on P0; later data is transferred on the same eight pins. External data transfer instructions which supply a sixteen-bit address, and any instruction accessing external program memory, output the high-order eight bits on P2 during the access cycle. (The 8031 *always* uses the pins of P0 and P2 for external addressing, but P1 and P3 are available for standard I/O.)

The eight pins of Port 3 (P3) each have a special function. Two external interrupts, two counter inputs, two serial data lines, and two timing control strobes use pins of P3 as described in Figure 6. Port 3 pins corresponding to functions not used are available for conventional I/O.

Even within a single port, I/O functions may be combined in many ways: input and output may be performed using different pins at the same time, or the same pins at different times; in parallel in some cases, and in serial in others; as test pins, or (in the case of Port 3) as additional special functions.



(MSB)				(LSB)			
RD	WR	T1	T0	INT1	INT0	TXD	RXD

#### Symbol Position Name and Significance

**RD** P3.7 Read data control output. Active low pulse generated by hardware when external data memory is read.

**WR** P3.6 Write data control output. Active low pulse generated by hardware when external data memory is written.

**T1** P3.5 Timer/counter 1 external input or test pin.

**T0** P3.4 Timer/counter 0 external input or test pin.

#### Symbol Position Name and Significance

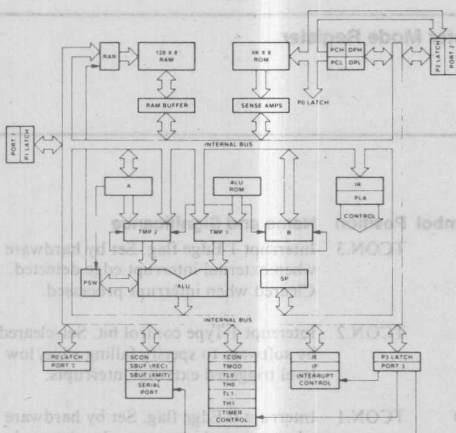
**INT1** P3.3 Interrupt 1 input pin. Low-level or falling-edge triggered.

**INT0** P3.2 Interrupt 0 input pin. Low-level or falling-edge triggered.

**TXD** P3.1 Transmit Data pin for serial port in UART mode. Clock output in shift register mode.

**RXD** P3.0 Receive Data pin for serial port in UART mode. Data I/O pin in shift register mode.

Figure 6. P3—Alternate Special Functions of Port 3



### Special Peripheral Functions

There are a few special features common among control-oriented computer systems:

- keeping track of elapsed real-time;
- maintaining a count of signal transitions;
- measuring the precise width of input pulses;
- communicating with other systems or people;
- closely monitoring asynchronous external events.

Until now, microprocessor systems needed peripheral chips such as timer/counters, USARTs, or interrupt controllers to meet these needs. The 8051 integrates all of these capabilities on-chip!

#### Timer/Counters

There are two sixteen-bit multiple-mode Timer/Counters on the 8051, each consisting of a "High" byte (corresponding to the 8048 "T" register) and a low byte (similar to the 8048 prescaler, with the additional flexibility of being

software-accessible). These registers are called, naturally enough, TH0, TL0, TH1, and TL1. Each pair may be independently software programmed to any of a dozen modes with a mode register designated TMOD (Figure 7), and controlled with register TCON (Figure 8).

The timer modes can be used to measure time intervals, determine pulse widths, or initiate events, with one-micro-second resolution, up to a maximum interval of 65,536 instruction cycles (over 65 milliseconds). Longer delays may easily be accumulated through software. Configured as a counter, the same hardware will accumulate external events at frequencies from D.C. to 500 KHz, with up to sixteen bits of precision.

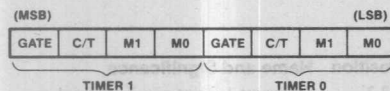
#### Serial Port Interface

Each microcomputer contains a high-speed, full-duplex, serial port which is software programmable to function in four basic modes: shift-register I/O expander, 8-bit UART, 9-bit UART, or interprocessor communications link. The UART modes will interface with standard I/O devices (e.g. CRTs, teletypewriters, or modems) at data rates from 122 baud to 31 kilobaud. Replacing the standard 12 MHz crystal with a 10.7 MHz crystal allows 110 baud. Even or odd parity (if desired) can be included with simple bit-handling software routines. Inter-processor communications in distributed systems takes place at 187 kilobaud with hardware for automatic address/data message recognition. Simple TTL or CMOS shift registers provide low-cost I/O expansion at a super-fast 1 Mega-baud. The serial port operating modes are controlled by the contents of register SCON (Figure 9).

#### Interrupt Capability and Control

(Interrupt capability is generally considered a CPU function. It is being introduced here since, from an applications point of view, interrupts relate more closely to peripheral and system interfacing.)



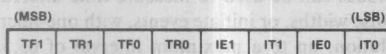


**GATE** Gating control. When set, Timer/counter "x" is enabled only while "INTx" pin is high and "TRx" control bit is set. When cleared, timer/counter is enabled whenever "TRx" control bit is set.

**C/T** Timer or Counter Selector. Cleared for Timer operation (input from internal system clock). Set for Counter operation (input from "Tx" input pin).

M1	M0	Operating Mode
0	0	MCS-48 Timer. "TLx" serves as five-bit prescaler.
0	1	16-bit timer/counter. "THx" and "TLx" are cascaded; there is no prescaler.
1	0	8-bit auto-reload timer/counter. "THx" holds a value which is to be reloaded into "TLx" each time it overflows.
1	1	(Timer 0) TL0 is an eight-bit timer/counter controlled by the standard Timer 0 control bits. TH0 is an eight-bit timer only controlled by Timer 1 control bits.
1	1	(Timer 1) Timer/counter 1 stopped.

**Figure 7. TMOD—Timer/Counter Mode Register**



**Symbol Position Name and Significance**

TF1	TCON.7	Timer 1 overflow Flag. Set by hardware on timer/counter overflow. Cleared when interrupt processed.
TR1	TCON.6	Timer 1 Run control bit. Set/cleared by software to turn timer/counter on/off.
TF0	TCON.5	Timer 0 overflow Flag. Set by hardware on timer/counter overflow. Cleared when interrupt processed.
TR0	TCON.4	Timer 0 Run control bit. Set/cleared by software to turn timer/counter on/off.

Symbol	Position	Name and Significance
IE1	TCON.3	Interrupt 1 Edge flag. Set by hardware when external interrupt edge detected. Cleared when interrupt processed.
IT1	TCON.2	Interrupt 1 Type control bit. Set/cleared by software to specify falling edge/low level triggered external interrupts.
IE0	TCON.1	Interrupt 0 Edge flag. Set by hardware when external interrupt edge detected. Cleared when interrupt processed.
IT0	TCON.0	Interrupt 0 Type control bit. Set/cleared by software to specify falling edge/low level triggered external interrupts.

**Figure 8. TCON—Timer/Counter Control/Status Register**

(MSB)				(LSB)			
SM0	SM1	SM2	REN	TB8	RB8	TI	RI

Symbol	Position	Name and Significance
SM0	SCON.7	Serial port Mode control bit 0. Set/cleared by software (see note).
SM1	SCON.6	Serial port Mode control bit 1. Set/cleared by software (see note).
SM2	SCON.5	Serial port Mode control bit 2. Set by software to disable reception of frames for which bit 8 is zero.
REN	SCON.4	Receiver Enable control bit. Set/cleared by software to enable/disable serial data reception.
TB8	SCON.3	Transmit Bit 8. Set/cleared by hardware to determine state of ninth data bit transmitted in 9-bit UART mode.

Symbol	Position	Name and Significance
RB8	SCON.2	Receive Bit 8. Set/cleared by hardware to indicate state of ninth data bit received.
TI	SCON.1	Transmit Interrupt flag. Set by hardware when byte transmitted. Cleared by software after servicing.
RI	SCON.0	Received Interrupt flag. Set by hardware when byte received. Cleared by software after servicing.

**Note—** the state of (SM0,SM1) selects:  
(0,0)—Shift register I/O expansion.  
(0,1)—8 bit UART, variable data rate.  
(1,0)—9 bit UART, fixed data rate.  
(1,1)—9 bit UART, variable data rate.

**Figure 9. SCON—Serial Port Control/Status Register**

These peripheral functions allow special hardware to monitor real-time signal interfacing without bothering the CPU. For example, imagine serial data is arriving from one CRT while being transmitted to another, and one timer/counter is tallying high-speed input transitions while the other measures input pulse widths. During all of this the CPU is thinking about something else.

But how does the CPU know when a reception, transmission, count, or pulse is finished? The 8051 programmer can choose from three approaches.

TCON and SCON contain status bits set by the hardware when a timer overflows or a serial port operation is completed. The first technique reads the control register into the accumulator, tests the appropriate bit, and does a conditional branch based on the result. This "polling" scheme (typically a three-instruction sequence though additional instructions to save and restore the accumulator may sometimes be needed) will surely be familiar to programmers used to multi-chip microcomputer systems and peripheral controller chips. This process is rather cumbersome, especially when monitoring multiple peripherals.

As a second approach, the 8051 can perform a conditional branch based on the state of any control or status bit or input pin in a single instruction; a four instruction sequence could poll the four simultaneous happenings mentioned above in just eight microseconds.

Unfortunately, the CPU must still drop what it's doing to test these bits. A manager cannot do his own work well if he is continuously monitoring his subordinates; they should interrupt him (or her) only when they need attention or guidance. So it is with machines: ideally, the CPU would not have to worry about the peripherals until they require servicing. At that time, it would postpone the

background task long enough to handle the appropriate device, then return to the point where it left off.

This is the basis of the third and generally optimal solution, hardware interrupts. The 8051 has five interrupt sources: one from the serial port when a transmission or reception is complete, two from the timers when overflows occur, and two from input pins INT0 and INT1. Each source may be independently enabled or disabled to allow polling on some sources or at some times, and each may be classified as high or low priority. A high priority source can interrupt a low priority service routine; the manager's boss can interrupt conferences with subordinates. These options are selected by the interrupt enable and priority control registers, IE and IP (Figures 10 and 11).

Each source has a particular program memory address associated with it (Table 3), starting at 0003H (as in the 8048) and continuing at eight-byte intervals. When an event enabled for interrupts occurs the CPU automatically executes an internal subroutine call to the corresponding address. A user subroutine starting at this location (or jumped to from this location) then performs the instructions to service that particular source. After completing the interrupt service routine, execution returns to the background program.

**Table 3. 8051 Interrupt Sources and Service Vectors**

Interrupt Source	Service Routine Starting Address
(Reset)	0000H
External 0	0003H
Timer/Counter 0	000BH
External 1	0013H
Timer/Counter 1	001BH
Serial Port	0023H

(MSB)							(LSB)						
EA	—	—	ES	ET1	EX1	ET0	EX0	—	—	—	—	—	—

Symbol	Position	Name and Significance
EA	IE.7	Enable All control bit. Cleared by software to disable all interrupts, independent of the state of IE.4-IE.0.
—	IE.6	(reserved)
—	IE.5	(reserved)
ES	IE.4	Enable Serial port control bit. Set/cleared by software to enable/disable interrupts from TI or RI flags.
ET1	IE.3	Enable Timer 1 control bit. Set/cleared by software to enable/disable interrupts from timer/counter 1.

Symbol	Position	Name and Significance
EX1	IE.2	Enable External interrupt 1 control bit. Set/cleared by software to enable/disable interrupts from INT1.
ET0	IE.1	Enable Timer 0 control bit. Set/cleared by software to enable/disable interrupts from timer/counter 0.
EX0	IE.0	Enable External interrupt 0 control bit. Set/cleared by software to enable/disable interrupts from INT0.

Figure 10. IE—Interrupt Enable Register

(MSB)							(LSB)						
—	—	—	PS	PT1	PX1	PT0	PX0	—	—	—	—	—	—

Symbol	Position	Name and Significance
—	IP.7	(reserved)
—	IP.6	(reserved)
—	IP.5	(reserved)
PS	IP.4	Serial port Priority control bit. Set/cleared by software to specify high/low priority interrupts for Serial port.
PT1	IP.3	Timer 1 Priority control bit. Set/cleared by software to specify high/low priority interrupts for timer/counter 1.

Symbol	Position	Name and Significance
PX1	IP.2	External interrupt 1 Priority control bit. Set/cleared by software to specify high/low priority interrupts for INT1.
PT0	IP.1	Timer 0 Priority control bit. Set/cleared by software to specify high/low priority interrupts for timer/counter 0.
PX0	IP.0	External interrupt 0 Priority control bit. Set/cleared by software to specify high/low priority interrupts for INT0.

Figure 11. IP—Interrupt Priority Control Register

Service Routine Starting Address	Interrupt Source
0000H	(Reserved)
0001H	External 0
0002H	Timer/Counter 0
0003H	External 1
0004H	Timer/Counter 1
0005H	Serial Port

Table 4. MCS-51™ Instruction Set Description

ARITHMETIC OPERATIONS				DATA TRANSFER (cont.)			
Mnemonic	Description	Byte	Cyc	Mnemonic	Description	Byte	Cyc
ADD A,Rn	Add register to Accumulator	1	1	MOVC A,@A+DPTR	Move Code byte relative to DPTR to A	1	2
ADD A,direct	Add direct byte to Accumulator	2	1	MOVC A,@A+PC	Move Code byte relative to PC to A	1	2
ADD A,@Ri	Add indirect RAM to Accumulator	1	1	MOVX A,@Ri	Move External RAM (8-bit addr) to A	1	2
ADD A,#data	Add immediate data to Accumulator	2	1	MOVX A,DPTR	Move External RAM (16-bit addr) to A	1	2
ADDC A,Rn	Add register to Accumulator with Carry	1	1	MOVX @Ri,A	Move A to External RAM (8-bit addr)	1	2
ADDC A,direct	Add direct byte to A with Carry flag	2	1	MOVX @DPTR,A	Move A to External RAM (16-bit addr)	1	2
ADDC A,@Ri	Add indirect RAM to A with Carry flag	1	1	PUSH direct	Push direct byte onto stack	2	2
ADDC A,#data	Add immediate data to A with Carry flag	2	1	POP direct	Pop direct byte from stack	2	2
SUBB A,Rn	Subtract register from A with Borrow	1	1	XCH A,Rn	Exchange register with Accumulator	1	1
SUBB A,direct	Subtract direct byte from A with Borrow	2	1	XCH A,direct	Exchange direct byte with Accumulator	2	1
SUBB A,@Ri	Subtract indirect RAM from A w Borrow	1	1	XCH A,@Ri	Exchange indirect RAM with A	1	1
SUBB A,#data	Subtract immed. data from A w Borrow	2	1	XCHD A,@Ri	Exchange low-order Digit ind. RAM w A	1	1
INC A	Increment Accumulator	1	1	BOOLEAN VARIABLE MANIPULATION			
INC Rn	Increment register	1	1	Mnemonic	Description	Byte	Cyc
INC direct	Increment direct byte	2	1	CLR C	Clear Carry flag	1	1
INC @Ri	Increment indirect RAM	1	1	CLR bit	Clear direct bit	2	1
DEC A	Decrement Accumulator	1	1	SETB C	Set Carry flag	1	1
DEC Rn	Decrement register	1	1	SETB bit	Set direct Bit	2	1
DEC direct	Decrement direct byte	2	1	CPL C	Complement Carry flag	1	1
DEC @Ri	Decrement indirect RAM	1	1	CPL bit	Complement direct bit	2	1
INC DPTR	Increment Data Pointer	1	2	ANL C,bit	AND direct bit to Carry flag	2	2
MUL AB	Multiply A & B	1	4	ANL C,bit	AND complement of direct bit to Carry	2	2
DIV AB	Divide A by B	1	4	ORL C,bit	OR direct bit to Carry flag	2	2
DA A	Decimal Adjust Accumulator	1	1	ORL C,bit	OR complement of direct bit to Carry	2	2
LOGICAL OPERATIONS				MOV C,bit	Move direct bit to Carry flag	2	1
Mnemonic	Description	Byte	Cyc	MOV bit,C	Move Carry flag to direct bit	2	2
ANL A,Rn	AND register to Accumulator	1	1	PROGRAM AND MACHINE CONTROL			
ANL A,direct	AND direct byte to Accumulator	2	1	Mnemonic	Description	Byte	Cyc
ANL A,@Ri	AND indirect RAM to Accumulator	1	1	ACALL addr11	Absolute Subroutine Call	2	2
ANL A,#data	AND immediate data to Accumulator	2	1	LCALL addr16	Long Subroutine Call	3	2
ANL direct,A	AND Accumulator to direct byte	2	1	RET	Return from subroutine	1	2
ANL direct,#data	AND immediate data to direct byte	3	2	RETI	Return from interrupt	1	2
ORL A,Rn	OR register to Accumulator	1	1	AJMP addr11	Absolute Jump	2	2
ORL A,direct	OR direct byte to Accumulator	2	1	LJMP addr16	Long Jump	3	2
ORL A,@Ri	OR indirect RAM to Accumulator	1	1	SJMP rel	Short Jump (relative addr)	2	2
ORL A,#data	OR immediate data to Accumulator	2	1	JMP @A+DPTR	Jump indirect relative to the DPTR	1	2
ORL direct,A	OR Accumulator to direct byte	2	1	JZ rel	Jump if Accumulator is Zero	2	2
ORL direct,#data	OR immediate data to direct byte	3	2	JNZ rel	Jump if Accumulator is Not Zero	2	2
XRL A,Rn	Exclusive-OR register to Accumulator	1	1	JC rel	Jump if Carry flag is set	2	2
XRL A,direct	Exclusive-OR direct byte to Accumulator	2	1	JNC rel	Jump if No Carry flag	2	2
XRL A,@Ri	Exclusive-OR indirect RAM to A	1	1	JB bit,rel	Jump if direct Bit set	3	2
XRL A,#data	Exclusive-OR immediate data to A	2	1	JNB bit,rel	Jump if direct Bit Not set	3	2
XRL direct,A	Exclusive-OR Accumulator to direct byte	2	1	JBC bit,rel	Jump if direct Bit is set & Clear bit	3	2
XRL direct,#data	Exclusive-OR immediate data to direct	3	2	CJNE A,direct,rel	Compare direct to A & Jump if Not Equal	3	2
CLR A	Clear Accumulator	1	1	CJNE A,#data,rel	Comp. immed. to A & Jump if Not Equal	3	2
CPL A	Complement Accumulator	1	1	CJNE Rn,#data,rel	Comp. immed. to reg. & Jump if Not Equal	3	2
RL A	Rotate Accumulator Left	1	1	CJNE @Ri,#data,rel	Comp. immed. to ind. & Jump if Not Equal	3	2
RLC A	Rotate A Left through the Carry flag	1	1	DJNZ Rn,rel	Decrement register & Jump if Not Zero	2	2
RR A	Rotate Accumulator Right	1	1	DJNZ direct,rel	Decrement direct & Jump if Not Zero	3	2
RRC A	Rotate A Right through Carry flag	1	1	NOP	No operation	1	1
SWAP A	Swap nibbles within the Accumulator	1	1	Notes on data addressing modes:			
DATA TRANSFER				Rn	Working register R0-R7		
Mnemonic	Description	Byte	Cyc	direct	128 internal RAM locations, any 1 I/O port, control or status register		
MOV A,Rn	Move register to Accumulator	1	1	@Ri	Indirect internal RAM location addressed by register R0 or R1		
MOV A,direct	Move direct byte to Accumulator	2	1	#data	8-bit constant included in instruction		
MOV A,@Ri	Move indirect RAM to Accumulator	1	1	#data16	16-bit constant included as bytes 2 & 3 of instruction		
MOV A,#data	Move immediate data to Accumulator	2	1	bit	128 software flags, any 1 I/O pin, control or status bit		
MOV Rn,A	Move Accumulator to register	1	1	Notes on program addressing modes:			
MOV Rn,direct	Move direct byte to register	2	2	addr16	Destination address for LCALL & LJMP may be anywhere within the 64-Kilobyte program memory address space.		
MOV Rn,#data	Move immediate data to register	2	1	addr11	Destination address for ACALL & AJMP will be within the same 2-Kilobyte page of program memory as the first byte of the following instruction.		
MOV direct,A	Move Accumulator to direct byte	2	1	rel	SJMP and all conditional jumps include an 8-bit offset byte. Range is +127-128 bytes relative to first byte of the following instruction.		
MOV direct,Rn	Move register to direct byte	2	2	All mnemonics copyrighted © Intel Corporation 1979			
MOV direct,direct	Move direct byte to direct	3	2				
MOV direct,@Ri	Move indirect RAM to direct byte	2	2				
MOV direct,#data	Move immediate data to direct byte	3	2				
MOV @Ri,A	Move Accumulator to indirect RAM	1	1				
MOV @Ri,direct	Move direct byte to indirect RAM	2	2				
MOV @Ri,#data	Move immediate data to indirect RAM	2	1				
MOV DPTR,#data16	Load Data Pointer with a 16-bit constant	3	2				

### 3. INSTRUCTION SET AND ADDRESSING MODES

The 8051 instruction set is extremely regular, in the sense that most instructions can operate with variables from several different physical or logical address spaces. Before getting deeply enmeshed in the instruction set proper, it is important to understand the details of the most common data addressing modes. Whereas Table 4 summarizes the instructions set broken down by functional

group, this chapter starts with the addressing mode classes and builds to include the related instructions.

#### Data Addressing Modes

MCS-51 assembly language instructions consist of an operation mnemonic and zero to three operands separated by commas. In two operand instructions the destination is specified first, then the source. Many byte-wide data



operations (such as ADD or MOV) inherently use the accumulator as a source operand and/or to receive the result. For the sake of clarity the letter "A" is specified in the source or destination field in all such instructions. For example, the instruction,

```
ADD A,<source>
```

will add the variable<source>to the accumulator, leaving the sum in the accumulator.

The operand designated "<source>" above may use any of four common logical addressing modes:

- Register—one of the working registers in the currently enabled bank.
- Direct—an internal RAM location, I/O port, or special-function register.
- Register-indirect—an internal RAM location, pointed to by a working register.
- Immediate data—an eight-bit constant incorporated into the instruction.

The first three modes provide access to the internal RAM and Hardware Register address spaces, and may therefore be used as source or destination operands; the last mode accesses program memory and may be a source operand only.

(It is hard to show a "typical application" of any instruction without involving instructions not yet described. The following descriptions use only the self-explanatory ADD and MOV instructions to demonstrate how the four addressing modes are specified and used. Subsequent examples will become increasingly complex.)

### Register Addressing

The 8051 programmer has access to eight "working registers," numbered R0-R7. The least-significant three-bits of the instruction opcode indicate one register within this logical address space. Thus, a function code and operand address can be combined to form a short (one byte) instruction (Figure 12.a).

The 8051 assembly language indicates register addressing with the symbol Rn (where n is from 0 to 7) or with a symbolic name previously defined as a register by the EQUate or SET directives. (For more information on assembler directives see the Macro Assembler Reference Manual.)

#### Example 1—Adding Two Registers Together

```
REGADR ADD CONTENTS OF REGISTER 1
      TO CONTENTS OF REGISTER 0

REGADR MOV A,R0
      ADD A,R1
      MOV R0,A
```

There are four such banks of working registers, only one of which is active at a time. Physically, they occupy the first 32 bytes of on-chip data RAM (addresses 0-1FH). PSW bits 4 and 3 determine which bank is active. A

hardware reset enables register bank 0; to select a different bank the programmer modifies PSW bits 4 and 3 accordingly.

#### Example 2—Selecting Alternate Memory Banks

```
MOV PSW,#00010000B SELECT BANK 2
```

Register addressing in the 8051 is the same as in the 8048 family, with two enhancements: there are four banks rather than one or two, and 16 instructions (rather than 12) can access them.

### Direct Byte Addressing

Direct addressing can access any on-chip variable or hardware register. An additional byte appended to the opcode specifies the location to be used (Figure 12.b).

Depending on the highest order bit of the direct address byte, one of two physical memory spaces is selected. When the direct address is between 0 and 127 (00H-7FH) one of the 128 low-order on-chip RAM locations is used. (Future microcomputers based on the MCS-51™ architecture may incorporate more than 128 bytes of on-chip RAM. Even if this is the case, only the low-order 128 bytes will be directly addressable. The remainder would be accessed indirectly or via the stack pointer.)

#### Example 3—Adding RAM Location Contents

```
DIRADR ADD CONTENTS OF RAM LOCATION 41H
      TO CONTENTS OF RAM LOCATION 40H

DIRADR MOV A,40H
      ADD A,41H
      MOV 40H,A
```

All I/O ports and special function, control, or status registers are assigned addresses between 128 and 255 (80H-0FFH). When the direct address byte is between these limits the corresponding hardware register is accessed. For example, Ports 0 and 1 are assigned direct addresses 80H and 90H, respectively. A complete list is presented in Table 5. Don't waste your time trying to memorize the addresses in Table 5. Since programs using absolute addresses for function registers would be difficult to write or understand, ASM51 allows and understands the abbreviations listed instead.

#### Example 4—Adding Input Port Data to Output Port Data

```
PRTADR ADD DATA INPUT ON PORT 1
      TO DATA PREVIOUSLY OUTPUT
      ON PORT 0

PRTADR MOV A,PO
      ADD A,P1
      MOV PO,A
```

Direct addressing allows all special-function registers in the 8051 to be read, written, or used as instruction operands. In general, this is the *only* method used for accessing I/O ports and special-function registers. If direct addressing is used with special-function register addresses other than those listed, the result of the instruction is undefined.

The 8048 does not have or need any generalized direct addressing mode, since there are only five special registers (BUS, P1, P2, PSW, & T) rather than twenty. Instead, 16 special 8048 opcodes control output bits or read or write each register to the accumulator. These functions are all subsumed by four of the 27 direct addressing instructions of the 8051.

**Table 5. 8051 Hardware Register Direct Addresses**

Register	Address	Function
P0	80H*	Port 0
SP	81H	Stack Pointer
DPL	82H	Data Pointer (Low)
DPH	83H	Data Pointer (High)
TCN	88H*	Timer register
TMOD	89H	Timer Mode register
TL0	8AH	Timer 0 Low byte
TL1	8BH	Timer 1 Low byte
TH0	8CH	Timer 0 High byte
TH1	8DH	Timer 1 High byte
P1	90H*	Port 1
SCON	98H*	Serial Port Control register
SBUF	99H	Serial Port data Buffer
P2	0A0H*	Port 2
IE	0A8H*	Interrupt Enable register
P3	0B0H*	Port 3
IP	0B8H*	Interrupt Priority register
PSW	0D0H*	Program Status Word
ACC	0E0H*	Accumulator (direct address)
B	0F0H*	B register

\* = bit addressable register.

### Register-Indirect Addressing

How can you handle variables whose locations in RAM are determined, computed, or modified while the program is running? This situation arises when manipulating sequential memory locations, indexed entries within tables in RAM, and multiple precision or string operations. Register or Direct addressing cannot be used, since their operand addresses are fixed at assembly time.

The 8051 solution is "register-indirect RAM addressing." R0 and R1 of each register bank may operate as index or pointer registers, their contents indicating an address into RAM. The internal RAM location so addressed is the actual operand used. The least significant bit of the instruction opcode determines which register is used as the "pointer" (Figure 12.c).

In the 8051 assembly language, register-indirect addressing is represented by a commercial "at" sign ("@" ) preceding R0, R1, or a symbol defined by the user to be equal to R0 or R1.

### Example 5—Indirect Addressing

```

INDADR ADD CONTENTS OF MEMORY LOCATION
        ADDRESSED BY REGISTER 1
        TO CONTENTS OF RAM LOCATION
        ADDRESSED BY REGISTER 0
INDADR MOV A, @R0
ADD A, @R1
MOV @R0, A

```

Indirect addressing on the 8051 is the same as in the 8048 family, except that all eight bits of the pointer register contents are significant; if the contents point to a non-existent memory location (i.e., an address greater than 7FH on the 8051) the result of the instruction is undefined. (Future microcomputers based on the MCS-51™ architecture could implement additional memory in the on-chip RAM logical address space at locations above 7FH.) The 8051 uses register-indirect addressing for five new instructions plus the 13 on the 8048.

### Immediate Addressing

When a source operand is a constant rather than a variable (i.e., the instruction uses a value known at assembly time), then the constant can be incorporated into the instruction. An additional instruction byte specifies the value used (Figure 12.d).

The value used is fixed at the time of ROM manufacture or EPROM programming and may not be altered during program execution. In the assembly language immediate operands are preceded by a number sign ("#"). The operand may be either a numeric string, a symbolic variable, or an arithmetic expression using constants.

### Example 6—Adding Constants Using Immediate Addressing

```

IMMADR ADD THE CONSTANT 12 (DECIMAL)
        TO THE CONSTANT 34 (DECIMAL)
        LEAVE SUM IN ACCUMULATOR
IMMADR MOV A, #12
ADD A, #34

```

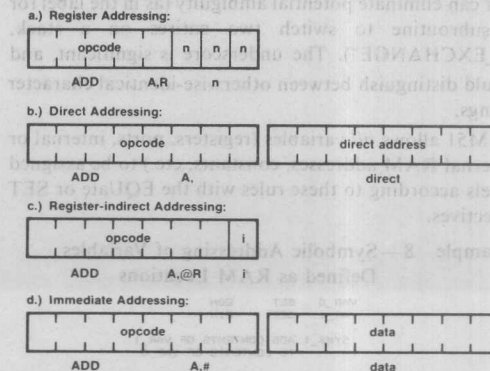
The preceding example was included for consistency; it has little practical value. Instead, ASM51 could compute the sum of two constants at assembly time.

### Example 7—Adding Constants Using ASM51 Capabilities

```

ASM51 LOAD ACC WITH THE SUM OF
        THE CONSTANT 12 (DECIMAL) AND
        THE CONSTANT 34 (DECIMAL)
ASM51 MOV A, # (12+34)

```



**Figure 12. Data Addressing Machine Code Formats**

## Addressing Mode Combinations

The above examples all demonstrated the use of the four data-addressing modes in two-operand instructions (MOV, ADD) which use the accumulator as one operand. The operations ADDC, SUBB, ANL, ORL, and XRL (all to be discussed later) could be substituted for ADD in each example. The first three modes may be also be used for the XCH operation or, in combination with the Immediate Addressing mode (and an additional byte), loaded with a constant. The one-operand instructions INC and DEC, DJNZ, and CJNE may all operate on the accumulator, or may specify the Register, Direct, and Register-indirect addressing modes. Exception: as in the 8048, DJNZ cannot use the accumulator or indirect addressing. (The PUSH and POP operations cannot inherently address the accumulator as a special register either. However, all three can *directly* address the accumulator as one of the twenty special-function registers by putting the symbol "ACC" in the operand field.)

## Advantages of Symbolic Addressing

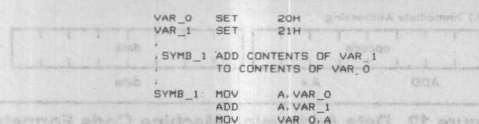
Like most assembly or higher-level programming languages, ASM51 allows instructions or variables to be given appropriate, user-defined symbolic names. This is done for instruction lines by putting a label followed by a colon (":") before the instruction proper, as in the above examples. Such symbols must start with an alphabetic character (remember what distinguished BACH from 0BACH?), and may include any combination of letters, numbers, question marks ("?",) and underscores ("\_"). For very long names only the first 31 characters are relevant.

Assembly language programs may intermix upper- and lower-case letters arbitrarily, but ASM51 converts both to upper-case. For example, ASM51 will internally process an "I" for an "i" and, of course, "A\_TOOTH" for "a\_tooth."

The underscore character makes symbols easier to read and can eliminate potential ambiguity (as in the label for a subroutine to switch two entires on a stack, "S\_EXCHANGE"). The underscore is significant, and would distinguish between otherwise-identical character strings.

ASM51 allows *all* variables (registers, ports, internal or external RAM addresses, constants, etc.) to be assigned labels according to these rules with the EQUate or SET directives.

Example 8—Symbolic Addressing of Variables Defined as RAM Locations



Notice from Table 4 that the MCS-51™ instruction set has relatively few instruction mnemonics (abbreviations) for the programmer to memorize. Different data types or addressing modes are determined by the operands specified, rather than variations on the mnemonic. For example, the mnemonic "MOV" is used by 18 different instructions to operate on three data types (bit, byte, and address). The fifteen versions which move byte variables between the logical address spaces are diagrammed in Figure 13. Each arrow shows the direction of transfer from source to destination.

Notice also that for most instructions allowing register addressing there is a corresponding direct addressing instruction and vice versa. This lets the programmer begin writing 8051 programs as if (s)he has access to 128 different registers. When the program has evolved to the point where the programmer has a fairly accurate idea how often each variable is used, he/she may allocate the working registers in each bank to the most "popular" variables. (The assembly cross-reference option will show exactly how often and where each symbol is referenced.) If symbolic addressing is used in writing the source program only the lines containing the symbol definition will need to be changed; the assembler will produce the appropriate instructions even though the rest of the program is left untouched. Editing only the first two lines of Example 8 will shrink the six-byte code segment produced in half.

How are instruction sets "counted"? There is no standard practice; different people assessing the same CPU using different conventions may arrive at different totals.

Each operation is then broken down according to the different addressing modes (or combinations of addressing modes) it can accommodate. The "CLR" mnemonic is used by two instructions with respect to bit variables ("CLR C" and "CLR bit") and once ("CLR A") with regards to bytes. This expansion yields the 111 separate instructions of Table 4.

The method used for the MCS-51\* instruction set first breaks it down into "operations": a basic function applied to a single data type. For example, the four versions of the ADD instruction are grouped to form one operation — addition of eight-bit variables. The six forms of the ANL instruction for byte variables make up a different operation; the two forms of ANL which operate on *bits* are considered still another. The MOV mnemonic is used by three different operation classes, depending on whether bit, byte, or 16-bit values are affected. Using this terminology the 8051 can perform 51 different operations.



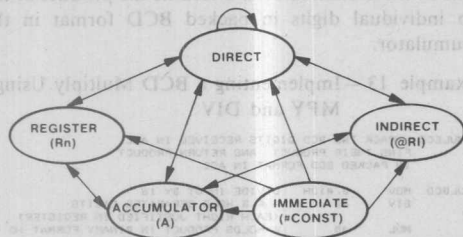


Figure 13. Road map for moving data bytes

Example 9—Redeclaring Example 8 Symbols as Registers

```

VAR_0 SET RO
VAR_1 SET R1
SYMB_2 ADD CONTENTS OF VAR_1
TO CONTENTS OF VAR_0
SYMB_2 MOV A, VAR_0
MOV VAR_0, A

```

Arithmetic Instruction Usage — ADD, ADDC, SUBB and DA

The ADD instruction adds a byte variable with the accumulator, leaving the result in the accumulator. The carry flag is set if there is an overflow from bit 7 and cleared otherwise. The AC flag is set to the carry-out from bit 3 for use by the DA instruction described later. ADDC adds the previous contents of the carry flag with the two byte variables, but otherwise is the same as ADD.

The SUBB (subtract with borrow) instruction subtracts the byte variable indicated and the contents of the carry flag together from the accumulator, and puts the result back in the accumulator. The carry flag serves as a "Borrow Required" flag during subtraction operations; when a greater value is subtracted from a lesser value (as in subtracting 5 from 1) requiring a borrow into the highest order bit, the carry flag is set; otherwise it is cleared.

When performing signed binary arithmetic, certain combinations of input variables can produce results which seem to violate the Laws of Mathematics. For example, adding 7FH (127) to itself produces a sum of 0FEH, which is the two's complement representation of -2 (refer back to Table 2)! In "normal" arithmetic, two positive values can't have a negative sum. Similarly, it is normally impossible to subtract a positive value from a negative value and leave a positive result — but in two's complement there are instances where this too may happen. Fundamentally, such anomalies occur when the magnitude of the resulting value is too great to "fit" into the seven bits allowed for it; there is no one-byte two's complement representation for 254, the true sum of 127 and 127.

The MCS-51™ processors detect whether these situations occur and indicate such errors with the OV flag. (OV may be tested with the conditional jump instructions JB and JNB, described under the Boolean Processor chapter.)

At a hardware level, OV is set if there is a carry out of bit 6 but not out of bit 7, or a carry out of bit 7 but not out of bit 6. When adding signed integers this indicates a negative number produced as the sum of two positive operands, or a positive sum from two negative operands; on SUBB this indicates a negative result after subtracting a negative number from a positive number, or a positive result when a positive number is subtracted from a negative number.

The ADDC and SUBB instructions incorporate the previous state of the carry (borrow) flag to allow multiple precision calculations by repeating the operation with successively higher-order operand bytes. In either case, the carry must be cleared before the first iteration.

If the input data for a multiple precision operation is an unsigned string of integers, upon completion the carry flag will be set if an overflow (for ADDC) or underflow (for SUBB) occurs. With two's complement signed data (i.e., if the most significant bit of the original input data indicates the sign of the string), the overflow flag will be set if overflow or underflow occurred.

Example 10—String Subtraction with Signed Overflow Detection

```

SUBSTR SUBTRACT STRING INDICATED BY R1
FROM STRING INDICATED BY R0 TO
PRECISION INDICATED BY R2.
CHECK FOR SIGNED UNDERFLOW WHEN DONE.

SUBSTR CLR C ;BORROW= 0.
MOV A, @R0
SUBB A, @R1 ;SUBTRACT NEXT PLACE
MOV @R0, A
INC R0 ;BUMP POINTERS
INC R1
DJNZ R2, SUBS1 ;LOOP AS NEEDED
WHEN DONE, TEST IF OVERFLOW OCCURRED
ON LAST ITERATION OF LOOP
JNB OV, OV_OK
; (OVERFLOW RECOVERY ROUTINE)
OV_OK RET ;RETURN

```

Decimal addition is possible by using the DA instruction in conjunction with ADD and/or ADDC. The eight-bit binary value in the accumulator resulting from an earlier addition of two variables (each a packed BCD digit-pair) is adjusted to form two BCD digits of four bits each. If the contents of accumulator bits 3-0 are greater than nine (xxxx1010-xxxx1111), or if the AC flag had been set, six is added to the accumulator producing the proper BCD digit in the low-order nibble. (This addition might itself set — but would not clear — the carry flag.) If the carry flag is set, or if the four high-order bits now exceed nine (1010xxxx-1111xxxx), these bits are incremented by six. The carry flag is left set if originally set or if either addition of six produces a carry out of the highest-order bit, indicating the sum of the original two BCD variables is greater than or equal to decimal 100.



### Example 11—Two Byte Decimal Add with Registers and Constants

```
BCDADD ADD THE CONSTANT 1,234 (DECIMAL) TO THE
        CONTENTS OF REGISTER PAIR (R3,R2)
        (ALREADY A 4 BCD-DIGIT VARIABLE)

BCDADD MOV A,R2
        ADD A,#34H
        DA A
        MOV R2,A
        ADDC A,R3
        DA A
        MOV R3,A
        RET
```

### Multiplication and Division

The instruction "MUL AB" multiplies the unsigned eight-bit integer values held in the accumulator and B-registers. The low-order byte of the sixteen-bit product is left in the accumulator, the higher-order byte in B. If the high-order eight-bits of the product are all zero the overflow flag is cleared; otherwise it is set. The programmer can poll OV to determine when the B register is non-zero and must be processed.

"DIV AB" divides the unsigned eight-bit integer in the accumulator by the unsigned eight-bit integer in the B-register. The integer part of the quotient is returned in the accumulator; the remainder in the B-register. If the B-register originally contained 00H then the overflow flag will be set to indicate a division error, and the values returned will be undefined. Otherwise OV is cleared.

The divide instruction is also useful for purposes such as radix conversion or separating bit fields of the accumulator. A short subroutine can convert an eight-bit unsigned binary integer in the accumulator (between 0 & 255) to a three-digit (two byte) BCD representation. The hundred's digit is returned in one register (HUND) and the ten's and one's digits returned as packed BCD in another (TENONE).

### Example 12—Use of DIV Instruction for Radix Conversion

```
BCD2BCD CONVERT 8-BIT BINARY VARIABLE IN ACC
        TO 3-DIGIT PACKED BCD FORMAT
        HUNDREDS' PLACE LEFT IN VARIABLE 'HUND',
        TENS' AND ONES' PLACES IN 'TENONE'

HUND EQU 21H
TENONE EQU 22H

BCD2BCD MOV B,#100 ;DIVIDE BY 100 TO DETERMINE NUMBER OF HUNDREDS
        DIV AB
        MOV HUND,A
        MOV A,#10 ;DIVIDE REMAINDER BY 10 TO DETERMINE # OF TENS LEFT
        XCH
        DIV AB
        SWAP A
        ADD A,B ;PACK BCD DIGITS IN ACC
        MOV TENONE,A
        RET
```

The divide instruction can also separate eight bits of data in the accumulator into sub-fields. For example, packed BCD data may be separated into two nibbles by dividing the data by 16, leaving the high-nibble in the accumulator and the low-order nibble (remainder) in B. The two digits may then be operated on individually or in conjunction with each other. This example receives two packed BCD

digits in the accumulator and returns the product of the two individual digits in packed BCD format in the accumulator.

### Example 13—Implementing a BCD Multiply Using MPY and DIV

```
MULBCD UNPACK TWO BCD DIGITS RECEIVED IN ACC,
        FIND THEIR PRODUCT, AND RETURN PRODUCT
        IN PACKED BCD FORMAT IN ACC

MULBCD MOV B,#10H ;DIVIDE INPUT BY 16
        DIV AB ;A & B HOLD SEPARATED DIGITS
        MUL AB ;EACH RIGHT JUSTIFIED IN REGISTER
        ;A HOLDS PRODUCT IN BINARY FORMAT (0 -
        ;99(DECIMAL) = 0 - 63H)
        MOV B,#10 ;DIVIDE PRODUCT BY 10
        DIV AB ;A HOLDS # OF TENS, B HOLDS REMAINDER
        SWAP A
        ORL A,B ;PACK DIGITS
        RET
```

### Logical Byte Operations — ANL, ORL, XRL

The instructions ANL, ORL, and XRL perform the logical functions AND, OR, and/or Exclusive-OR on the two byte variables indicated, leaving the results in the first. No flags are affected. (A word to the wise — do not vocalize the first two mnemonics in mixed company.)

These operations may use all the same addressing modes as the arithmetics (ADD, etc.) but unlike the arithmetics, they are not restricted to operating on the accumulator. Directly addressed bytes may be used as the destination with either the accumulator or a constant as the source. These instructions are useful for clearing (ANL), setting (ORL), or complementing (XRL) one or more bits in a RAM, output ports, or control registers. The pattern of bits to be affected is indicated by a suitable mask byte. Use immediate addressing when the pattern to be affected is known at assembly time (Figure 14); use the accumulator versions when the pattern is computed at run-time.

I/O ports are often used for parallel data in formats other than simple eight-bit bytes. For example, the low-order five bits of port 1 may output an alphabetic character code (hopefully) without disturbing bits 7-5. This can be a simple two-step process. First, clear the low-order five pins with an ANL instruction; then set those pins corresponding to ones in the accumulator. (This example assumes the three high-order bits of the accumulator are originally zero.)

### Example 14—Reconfiguring Port Size with Logical Byte Instructions

```
OUT_PX ANL P1,#11100000B ;CLEAR BITS P1.4 - P1.0
        ORL P1,A ;SET P1 PINS CORRESPONDING TO SET ACC
        ;BITS
        RET
```

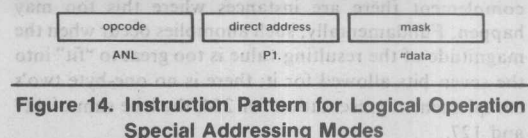


Figure 14. Instruction Pattern for Logical Operation Special Addressing Modes

In this example, low-order bits remaining high may “glitch” low for one machine cycle. If this is undesirable, use a slightly different approach. First, set all pins corresponding to accumulator one bits, then clear the pins corresponding to zeroes in low-order accumulator bits. Not all bits will change from original to final state at the same instant, but no bit makes an intermediate transition.

#### Example 15—Reconfiguring I/O Port Size without Glitching

```
ALT_PX: ORL P1,A
        ORL A,#1100000B
        ANL P1,A
        RET
```

#### Program Control — Jumps, Calls, Returns

Whereas the 8048 only has a single form of the simple jump instruction, the 8051 has three. Each causes the program to unconditionally jump to some other address. They differ in how the machine code represents the destination address.

LJMP (Long Jump) encodes a sixteen-bit address in the second and third instruction bytes (Figure 15.a); the destination may be anywhere in the 64 Kilobyte program memory address space.

The two-byte AJMP (Absolute Jump) instruction encodes its destination using the same format as the 8048: address bits 10 through 8 form a three bit field in the opcode and address bits 7 through 0 form the second byte (Figure 15.b). Address bits 15-12 are unchanged from the (incremented) contents of the P.C., so AJMP can only be used when the destination is known to be within the same 2K memory block. (Otherwise ASM51 will point out the error.)

A different two-byte jump instruction is legal with any nearby destination, regardless of memory block boundaries or “pages.” SJMP (Short Jump) encodes the destination with a program counter-relative address in the second byte (Figure 15.c). The CPU calculates the

destination at run-time by adding the signed eight-bit displacement value to the incremented P.C. Negative offset values will cause jumps up to 128 bytes backwards; positive values up to 127 bytes forwards. (SJMP with 00H in the machine code offset byte will proceed with the following instruction).

In keeping with the 8051 assembly language goal of minimizing the number of instruction mnemonics, there is a “generic” form of the three jump instructions. ASM51 recognizes the mnemonic JMP as a “pseudo-instruction,” translating it into the machine instructions LJMP, AJMP, or SJMP, depending on the destination address.

Like SJMP, all conditional jump instructions use relative addressing. JZ (Jump if Zero) and JNZ (Jump if Not Zero) monitor the state of the accumulator as implied by their names, while JC (Jump on Carry) and JNC (Jump on No Carry) test whether or not the carry flag is set. All four are two-byte instructions, with the same format as Figure 15.c. JB (Jump on Bit), JNB (Jump on No Bit) and JBC (Jump on Bit then Clear Bit) can test any status bit or input pin with a three byte instruction; the second byte specifies which bit to test and the third gives the relative offset value.

There are two subroutine-call instructions, LCALL (Long Call) and ACALL (Absolute Call). Each increments the P.C. to the first byte of the following instruction, then pushes it onto the stack (low byte first). Saving both bytes increments the stack pointer by two. The subroutine’s starting address is encoded in the same ways as LJMP and AJMP. The generic form of the call operation is the mnemonic CALL, which ASM51 will translate into LCALL or ACALL as appropriate.

The return instruction RET pops the high- and low-order bytes of the program counter successively from the stack, decrementing the stack pointer by two. Program execution continues at the address previously pushed; the first byte of the instruction immediately following the call.

When an interrupt request is recognized by the 8051 hardware, two things happen. Program control is automatically “vectored” to one of the interrupt service routine starting addresses by, in effect, forcing the CPU to process an LCALL instead of the next instruction. This automatically stores the return address on the stack. (Unlike the 8048, no status information is automatically saved.)

Secondly, the interrupt logic is disabled from accepting any other interrupts from the same or lower priority. After completing the interrupt service routine, executing an RETI (Return from Interrupt) instruction will return execution to the point where the background program was interrupted — just like RET — while restoring the interrupt logic to its previous state.

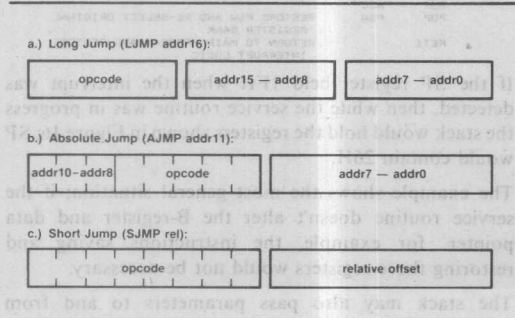


Figure 15. Jump Instruction Machine Code Formats

## Operate-and-branch instructions — CJNE, DJNZ

Two groups of instructions combine a byte operation with a conditional jump based on the results.

CJNE (Compare and Jump if Not Equal) compares two byte operands and executes a jump if they disagree. The carry flag is set following the rules for subtraction: if the unsigned integer value of the first operand is less than that of the second it is set; otherwise, it is cleared. However, neither operand is modified.

The CJNE instruction provides, in effect, a one-instruction "case" statement. This instruction may be executed repeatedly, comparing the code variable to a list of "special case" value: the code segment following the instruction (up to the destination label) will be executed only if the operands match. Comparing the accumulator or a register to a series of constants is a convenient way to check for special handling or error conditions; if none of the cases match the program will continue with "normal" processing.

A typical example might be a word processing device which receives ASCII characters through the serial port and drives a thermal hard-copy printer. A standard routine translates "printing" characters to bit patterns, but control characters (<DEL>, <CR>, <LF>, <BEL>, <ESC> or <SP>) must invoke corresponding special routines. Any other character with an ASCII code less than 20H should be translated into the <NUL> value, 00H, and processed with the printing characters.

### Example 16—Case Statements Using CJNE

```
CHAR EQU R7 ; CHARACTER CODE VARIABLE
INTERR: CJNE CHAR, #7FH, INTP_1 ; (SPECIAL ROUTINE FOR RUBOUT CODE)
        RET
INTP_1: CJNE CHAR, #07H, INTP_2 ; (SPECIAL ROUTINE FOR BELL CODE)
        RET
INTP_2: CJNE CHAR, #0AH, INTP_3 ; (SPECIAL ROUTINE FOR LFEE CODE)
        RET
INTP_3: CJNE CHAR, #0DH, INTP_4 ; (SPECIAL ROUTINE FOR RETURN CODE)
        RET
INTP_4: CJNE CHAR, #1BH, INTP_5 ; (SPECIAL ROUTINE FOR ESCAPE CODE)
        RET
INTP_5: CJNE CHAR, #20H, INTP_6 ; (SPECIAL ROUTINE FOR SPACE CODE)
        RET
INTP_6: JC PRINTC ; JUMP IF CODE > 20H
        MOV CHAR, #0 ; REPLACE CONTROL CHARACTERS WITH
        ; NULL CODE
PRINTC: ; PROCESS STANDARD PRINTING
        ; CHARACTER
        RET
```

DJNZ (Decrement and Jump if Not Zero) decrements the register or direct address indicated and jumps if the result is not zero, without affecting any flags. This provides a simple means for executing a program loop a given number of times, or for adding a moderate time delay (from 2 to 512 machine cycles) with a single instruction. For example, a 99-usec. software delay loop can be added to code forcing an I/O pin low with only two instructions.

### Example 17—Inserting a Software Delay with DJNZ

```
CLR WR
MOV R2, #49
DJNZ R2, $
SETB WR
```

The dollar sign in this example is a special character meaning "the address of this instruction." It is useful in eliminating instruction labels on the same or adjacent source lines. CJNE and DJNZ (like all conditional jumps) use program-counter relative addressing for the destination address.

## Stack Operations — PUSH, POP

The PUSH instruction increments the stack pointer by one, then transfers the contents of the single byte variable indicated (direct addressing only) into the internal RAM location addressed by the stack pointer. Conversely, POP copies the contents of the internal RAM location addressed by the stack pointer to the byte variable indicated, then decrements the stack pointer by one.

(Stack Addressing follows the same rules, and addresses the same locations as Register-indirect. Future microcomputers based on the MCS-51™ CPU could have up to 256 bytes of RAM for the stack.)

Interrupt service routines must not change any variable or hardware registers modified by the main program, or else the program may not resume correctly. (Such a change might look like a spontaneous random error.) Resources used or altered by the service routine (Accumulator, PSW, etc.) must be saved and restored to their previous value before returning from the service routine. PUSH and POP provide an efficient and convenient way to save register states on the stack.

### Example 18—Use of the Stack for Status Saving on Interrupts

```
LOC_TMP EQU $ ; REMEMBER LOCATION COUNTER

ORG 0003H ; STARTING ADDRESS FOR INTERRUPT ROUTINE
LAMP: JUMP SERVER ; JUMP TO ACTUAL SERVICE ROUTINE LOCATED
        ; ELSEWHERE

SERVER: ORG LOC_TMP ; RESTORE LOCATION COUNTER
        PUSH PSW ; SAVE ACCUMULATOR (NOTE DIRECT ADDRESSING
        ; NOTATION)
        PUSH B ; SAVE B REGISTER
        PUSH DPL ; SAVE DATA POINTER
        MOV PSW, #00001000B ; SELECT REGISTER BANK 1

        POP DPH ; RESTORE REGISTERS IN REVERSE ORDER
        POP DPL
        POP B
        POP ACC
        POP PSW ; RESTORE PSW AND RE-SELECT ORIGINAL
        ; REGISTER BANK
        RETI ; RETURN TO MAIN PROGRAM AND RESTORE
        ; INTERRUPT LOGIC
```

If the SP register held 1FH when the interrupt was detected, then while the service routine was in progress the stack would hold the registers shown in Figure 16; SP would contain 26H.

The example shows the most general situation; if the service routine doesn't alter the B-register and data pointer, for example, the instructions saving and restoring those registers would not be necessary.

The stack may also pass parameters to and from subroutines. The subroutine can indirectly address the parameters derived from the contents of the stack pointer.



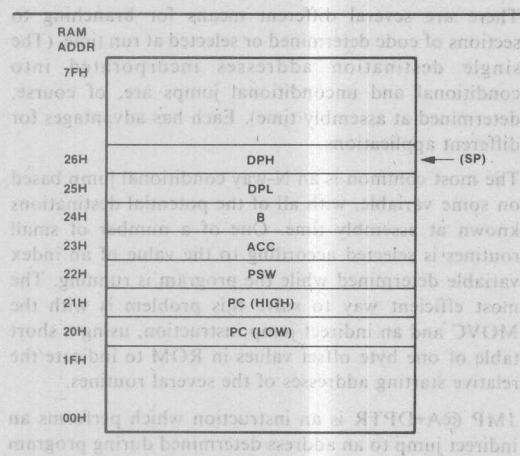


Figure 16. Stack contents during interrupt

One advantage here is simplicity. Variables need not be allocated for specific parameters, a potentially large number of parameters may be passed, and different calling programs may use different techniques for determining or handling the variables.

For example, the following subroutine reads out a parameter stored on the stack by the calling program, uses the low order bits to access a local look-up table holding bit patterns for driving the coils of a four phase stepper motor, and stores the appropriate bit pattern back in the same position on the stack before returning. The accumulator contents are left unchanged.

Example 19—Passing Variable Parameters to Subroutines Using the Stack

```

NEXTPOS MOV RO, SP      ; ACCESS LOCATION PARAMETER PUSHED INTO
DEC RO      ; ACCESS LOCATION PARAMETER PUSHED INTO
DEC RO      ; ACCESS LOCATION PARAMETER PUSHED INTO
XCH A, @RO   ; READ INPUT PARAMETER AND SAVE
ANL A, #03H  ; MASK ALL BUT LOW-ORDER TWO BITS
ADD A, #2    ; ALLOW FOR OFFSET FROM MOVX TO TABLE
MOVX A, @A+PC ; READ LOOK-UP TABLE ENTRY
XCH A, @RO   ; PASS BACK TRANSLATED VALUE AND RESTORE
; ACC
RET          ; RETURN TO BACKGROUND PROGRAM
STPTBL: DB 01101111B ; POSITION 0
DB 01011111B ; POSITION 1
DB 01111111B ; POSITION 2
DB 10101111B ; POSITION 3

```

The background program may reach this subroutine with several different calling sequences, all of which PUSH a value before calling the routine and POP the result after. A motor on Port 1 may be initialized by placing the desired position (zero) on the stack before calling the subroutine and outputting the results directly to a port afterwards.

Example 20—Sending and Receiving Data Parameters Via the Stack

```

CLR A
PUSH ACC
CALL NXTPOS
POP P1

```

If the position of the motor is determined by the contents of variable POSM1 (a byte in internal RAM) and the position of a second motor on Port 2 is determined by the data input to the low-order nibble of Port 2, a six-instruction sequence could update them both.

Example 21—Loading and Unloading Stack Direct from I/O Ports

```

POSM1 EQU 51H
PUSH POSM1
CALL NXTPOS
POP P1
PUSH P2
CALL NXTPOS
POP P2

```

## Data Pointer and Table Look-up instructions — MOV, INC, MOVC, JMP

The data pointer can be loaded with a 16-bit value using the instruction MOV DPTR, #data16. The data used is stored in the second and third instruction bytes, high-order byte first. The data pointer is incremented by INC DPTR. A 16-bit increment is performed; an overflow from the low byte will carry into the high-order byte. Neither instruction affects any flags.

The MOVC (Move Constant) instructions (MOVC A, @A+DPTR and MOVC A, @A+PC) read into the accumulator bytes of data from the program memory logical address space. Both use a form of indexed addressing: the former adds the unsigned eight-bit accumulator contents with the sixteen-bit data pointer register, and uses the resulting sum as the address from which the byte is fetched. A sixteen-bit addition is performed; a carry-out from the low-order eight bits may propagate through higher-order bits, but the contents of the DPTR are not altered. The latter form uses the incremented program counter as the "base" value instead of the DPTR (figure 17). Again, neither version affects the flags.

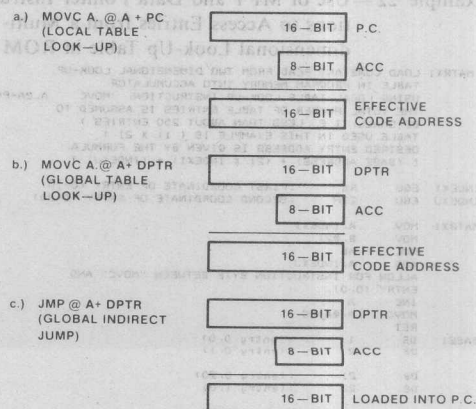


Figure 17. Operation of MOVC instructions



Each can be part of a three step sequence to access look-up tables in ROM. To use the DPTR-relative version, load the Data Pointer with the starting address of a look-up table; load the accumulator with (or compute) the index of the entry desired; and execute `MOVC A,@A+DPTR`. Unlike the similar `MOVP3` instructions in the 8048, the table may be located anywhere in program memory. The data pointer may be loaded with a constant for short tables. Or to allow more complicated data structures, or tables with more than 256 entries, the values for DPH and DPL may be computed or modified with the standard arithmetic instruction set.

The PC-relative version has the advantage of not affecting the data pointer. Again, a look-up sequence takes three steps: load the accumulator with the index; compensate for the offset from the look-up instruction to the start of the table by adding the number of bytes separating them to the accumulator; then execute the `MOVC A,@A+PC` instruction.

Let's look at a non-trivial situation where this instruction would be used. Some applications store large multi-dimensional look-up tables of dot matrix patterns, non-linear calibration parameters, and so on in a linear (one-dimensional) vector in program memory. To retrieve data from the tables, variables representing matrix indices must be converted to the desired entry's memory address. For a matrix of dimensions (MDIMEN x NDIMEN) starting at address BASE and respective indices INDEXI and INDEXJ, the address of element (INDEXI, INDEXJ) is determined by the formula:

$$\text{Entry Address} = \text{BASE} + (\text{NDIMEN} \times \text{INDEXI}) + \text{INDEXJ}$$

The code shown below can access any array with less than 255 entries (i.e., an 11x21 array with 231 elements). The table entries are defined using the Data Byte ("DB") directive, and will be contained in the assembly object code as part of the accessing subroutine itself.

#### Example 22—Use of MPY and Data Pointer Instructions to Access Entries from a Multi-dimensional Look-Up Table in ROM

```

;MATRIX1 LOAD CONSTANT READ FROM TWO DIMENSIONAL LOOK-UP
;TABLE IN PROGRAM MEMORY INTO ACCUMULATOR
;USING LOCAL TABLE LOOK-UP INSTRUCTION. 'MOVC A,@A+PC
;THE TOTAL NUMBER OF TABLE ENTRIES IS ASSUMED TO
;BE SMALL, I.E. LESS THAN ABOUT 250 ENTRIES )
;TABLE USED IN THIS EXAMPLE IS ( 11 X 21 )
;DESIRED ENTRY ADDRESS IS GIVEN BY THE FORMULA,
;( BASE ADDRESS ) + ( 21 X INDEXI ) + ( INDEXJ )
;
INDEXI EQU R6 ;FIRST COORDINATE OF ENTRY (0-10)
INDEXJ EQU R7 ;SECOND COORDINATE OF ENTRY (0-20)
;
MATRIX1: MOV A,INDEXI
;
MOV B,#21
MUL AB
ADD A,INDEXJ
;
;ALLOW FOR INSTRUCTION BYTE BETWEEN "MOVC" AND
;ENTRY (0,0)
INC A
MOVC A,@A+PC
RET
;
BASE1: DB 1 ;entry (0,0)
DB 2 ;entry (0,1)
;
;
DB 21 ;entry (0,20)
DB 22 ;entry (1,0)
;
;
DB 42 ;entry (1,20)
;
;
DB 231 ;entry (10,20)

```

There are several different means for branching to sections of code determined or selected at run time. (The single destination addresses incorporated into conditional and unconditional jumps are, of course, determined at assembly time). Each has advantages for different applications.

The most common is an N-way conditional jump based on some variable, with all of the potential destinations known at assembly time. One of a number of small routines is selected according to the value of an index variable determined while the program is running. The most efficient way to solve this problem is with the `MOVC` and an indirect jump instruction, using a short table of one byte offset values in ROM to indicate the relative starting addresses of the several routines.

`JMP @A+DPTR` is an instruction which performs an indirect jump to an address determined during program execution. The instruction adds the eight-bit unsigned accumulator contents with the contents of the sixteen-bit data pointer, just like `MOVC A,@A+DPTR`. The resulting sum is loaded into the program counter and is used as the address for subsequent instruction fetches. Again, a sixteen-bit addition is performed; a carry out from the low-order eight bits may propagate through the higher-order bits. In this case, neither the accumulator contents nor the data pointer is altered.

The example subroutine below reads a byte of RAM into the accumulator from one of four alternate address spaces, as selected by the contents of the variable `MEMSEL`. The address of the byte to be read is determined by the contents of `R0` (and optionally `R1`). It might find use in a printing terminal application, where four different model printers all use the same ROM code but use different types and sizes of buffer memory for different speeds and options.

#### Example 23—N-Way Branch and Computed Jump Instructions via `JMP @ADPTR`

```

MEMSEL EQU R3
JUMP_4: MOV A, MEMSEL
MOV DPTR, #JMP_TBL
MOVC A,@ADPTR
JMP A, DPTR
;
JMP_TBL: DB MEMSP0-JMP_TBL
DB MEMSP1-JMP_TBL
DB MEMSP2-JMP_TBL
DB MEMSP3-JMP_TBL
;
MEMSP0: MOV A,R0 ;READ FROM INTERNAL RAM
RET
MEMSP1: MOVX A,@R0 ;READ FROM 256 BYTES OF EXTERNAL RAM
RET
MEMSP2: MOV DPH,R1
MOVX A,@DPTR ;READ FROM 64K BYTES OF EXTERNAL RAM
RET
MEMSP3: MOV A,R1
ANL A,#07H
ORL P1,#1111000B
ORL P1,A
MOVX A,@R0 ;READ FROM 4K BYTES OF EXTERNAL RAM
RET

```

Note that this approach is suitable whenever the size of jump table plus the length of the alternate routines is less than 256 bytes. The jump table and routines may be located anywhere in program memory, independent of 256-byte program memory pages.

Example 24 — N-Way Branch with 128 Optional Destinations

W29		B0	B1	Destinations				B2	B3	B4	B5	B6	B7	H000
OPTION	EGU	R3												
JMP12B	MOV	A, OPTION		MULTIPLY BY 2 FOR 2 BYTE JUMP TABLE										
	RL	DPR, INSTBL		FIRST ENTRY IN JUMP TABLE										
	MOV	Q#>DPR		JUMP INTO JUMP TABLE										
INSTBL	AJMP	PROCOO		12B CONSECUTIVE										
	AJMP	PRDC01		AJMP INSTRUCTIONS										
	AJMP	PRDC02												
	AJMP	PRC7E												
	AJMP	PRC7F												

In those rare situations where even 128 options are insufficient, or where the destination routines may cross a 2K page boundary, the above approach may be modified slightly as shown below.

### Example 25—256-Way Branch Using Address Look-Up Tables

```

RTMP EQU R7
JMP256 MOV DPTR, #ADRTBL ; FIRST ENTRY IN TABLE OF ADDRESSES
        MOV A, OPTION
        CLR C
        RLC A ; MULTIPLY BY 2 FOR 2 BYTE JUMP TABLE
        JNC LOW12B
        INC DPH
LOW12B MOV RTMP, A ; SAVE ACC FOR HIGH BYTE READ
        MOVX A, A, #DPTR ; READ LOW BYTE FROM JUMP TABLE
        INC A
        MOVX A, A, #DPTR ; GET LOW-ORDER BYTE FROM TABLE
        PUSH ACC
        MOV A, RTMP
        MOVX A, A, #DPTR ; GET HIGH-ORDER BYTE FROM TABLE
        PUSH ACC
        THE TWO ACC PUSHES HAVE PRODUCED
        A "RETURN ADDRESS" ON THE STACK WHICH CORRESPONDS
        TO THE DESIRED STARTING ADDRESS
        IT MAY BE REACHED BY POPPING THE STACK
        INTO THE PC
        RET
ADRTBL DW PROCOO ; UP TO 256 CONSECUTIVE DATA
        DW PROCO1 ; WORDS INDICATING STARTING ADDRESSES
        DW PROCCF
        DUMMY CODE ADDRESS DEFINITIONS NEEDED BY ABOVE
        TWO EXAMPLES:
PROCOO NOP
PROCO1 NOP
PROCO2 NOP
PROCFE NOP
PROCFE NOP
PROCCF NOP

```

The commonly accepted terms for tasks at either end of the computational vs. control application spectrum are, respectively, "number-crunching" and "bit-banging".

### Direct Bit Addressing

As in direct-byte addressing, bit 7 of the address byte switches between two physical address spaces. Values between 0 and 127 (00H-7FH) define bits in internal RAM locations 20H to 2FH (Figure 18a); address bytes between 128 and 255 (80H-0FFH) define bits in the 2 x “special-function” register address space (Figure 18b). If no 2 x “special-function” register corresponds to the direct bit address used the result of the instruction is undefined.

Bits so addressed have many wondrous properties. They may be set, cleared, or complemented with the two byte instructions SETB, CLR, or CPL. Bits may be moved to and from the carry flag with MOV. The logical AND and ORL functions may be performed between the carry and either the addressed bit or its complement.

The “MOV” mnemonic can be used to load an addressable bit into the carry flag (“MOV C, bit”) or to copy the state of the carry to such a bit (“MOV bit, C”). These instructions are often used for implementing serial I/O algorithms via software or to adapt the standard I/O port structure.

It is sometimes desirable to "re-arrange" the order of I/O pins because of considerations in laying out printed circuit boards. When interfacing the 8051 to an immediately adjacent device with "weighted" input pins, such as keyboard column decoder, the corresponding pins are likely to be not aligned (Figure 19).

There is a trade-off in "scrambling" the interconnections with either interwoven circuit board traces or through software. This is extremely cumbersome (if not impossible) to do with byte-oriented computer architectures. The 8051's unique set of Boolean instructions makes it simple to move individual bits between arbitrary locations.

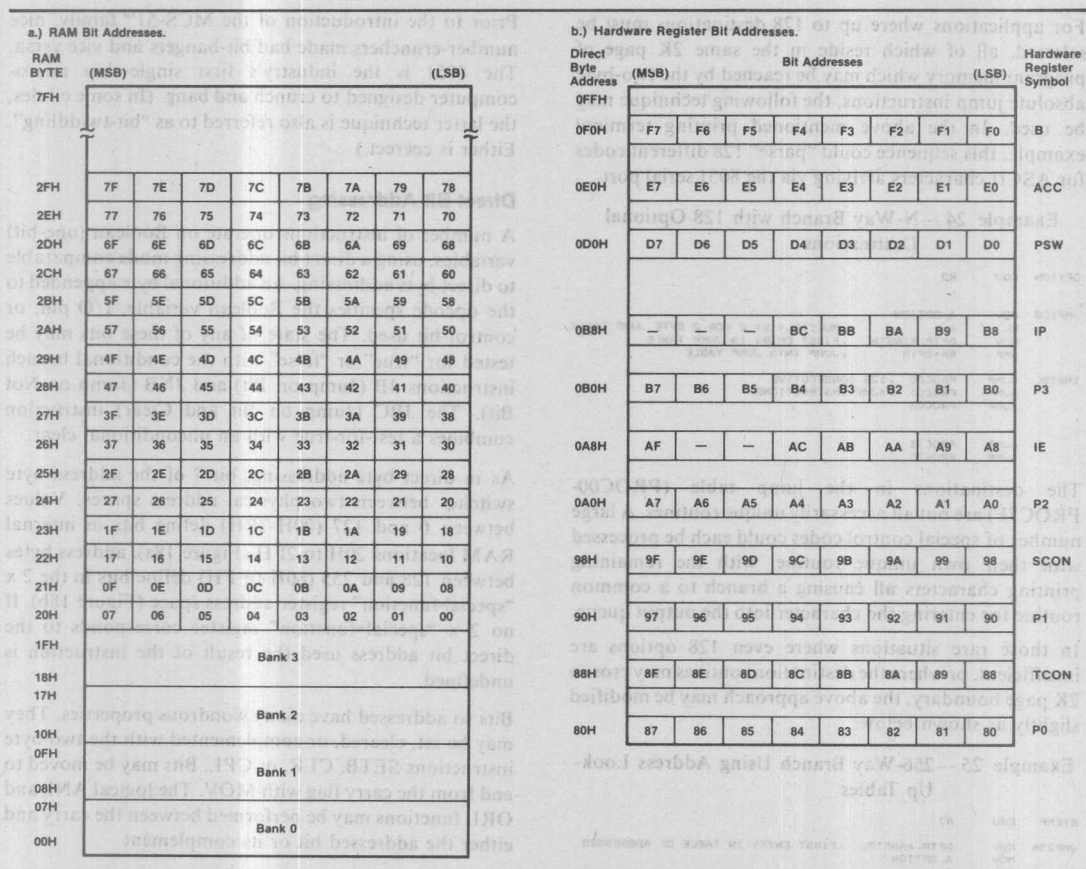


Figure 18. Bit Address Maps

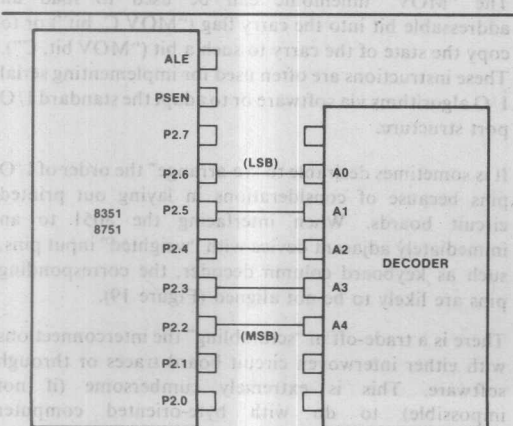


Figure 19. "Mismatch" Between I/O port and Decoder

Example 26—Re-ordering I/O Port Configuration

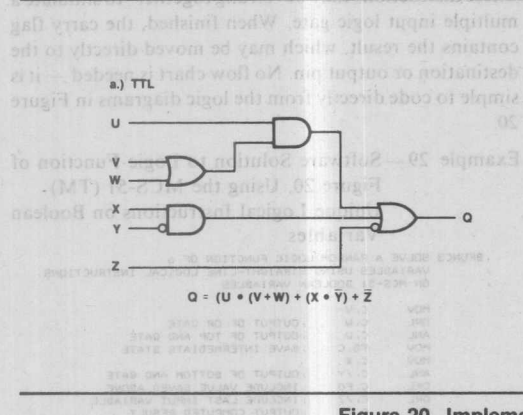
```

OUT_PZ  RRC  A      ;MOVE ORIGINAL ACC 0 INTO CY
MOV      P2 6.C  ;STORE CARRY TO PIN P26
RRC      A      ;MOVE ORIGINAL ACC 1 INTO CY
MOV      P2 5.C  ;STORE CARRY TO PIN P25
RRC      A      ;MOVE ORIGINAL ACC 2 INTO CY
MOV      P2 4.C  ;STORE CARRY TO PIN P24
RRC      A      ;MOVE ORIGINAL ACC 3 INTO CY
MOV      P2 3.C  ;STORE CARRY TO PIN P23
RRC      A      ;MOVE ORIGINAL ACC 4 INTO CY
MOV      P2 2.C  ;STORE CARRY TO PIN P22
RET

```

### Solving Combinatorial Logic Equations — ANL, ORL

Virtually all hardware designers are familiar with the problem of solving complex functions using combinatorial logic. The technologies involved may vary greatly, from multiple contact relay logic, vacuum tubes, TTL, or CMOS to more esoteric approaches like fluidics, but in each case the goal is the same; a Boolean (true/false) function is computed on a number of Boolean variables.



b.) Relay logic.

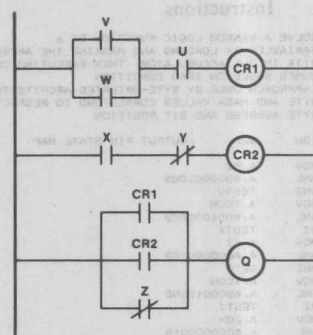


Figure 20. Implementations of Boolean functions

Figure 20 shows the logic diagram for an arbitrary function of six variables named U through Z using standard logic and relay logic symbols. Each is a solution of the equation,

$$Q = (U \cdot (V + W)) + (X \cdot Y) + Z$$

(While this equation could be reduced using Karnaugh Maps or algebraic techniques, that is not the purpose of this example. Even a minor change to the function equation would require re-reducing from scratch.)

Most digital computers can solve equations of this type with standard word-wide logical instructions and conditional jumps. Still, such software solutions seem somewhat sloppy because of the many paths through the program the computation can take.

Assume U and V are input pins being read by different input ports, W and X are status bits for two peripheral controllers (read as I/O ports), and Y and Z are software flags set or cleared earlier in the program. The end result must be written to an output pin on some third port.

For the sake of comparison we will implement this function with software drawn from three proper subsets of the MCS-51™ instruction set. The first two implementations follow the flow chart shown in Figure 21. Program flow would embark on a route down a test-and-branch tree and leaves either the "True" or "Not True" exit ASAP. These exits then write the output port with the data previously written to the same port with the result bit respectively one or zero.

In the first case, we assume there are no instructions for addressing individual bits other than special flags like the carry. This is typical of many older microprocessors and mainframe computers designed for number-crunching. MCS-51™ mnemonics are used here, though for most other machines the issue would be even further clouded by their use of operation-specific mnemonics like

INPUT, OUTPUT, LOAD, STORE, etc., instead of the universal MOV.

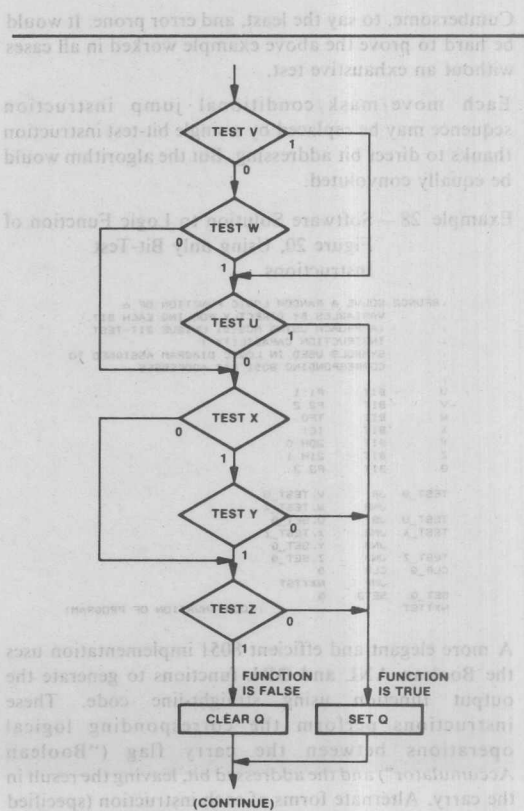


Figure 21. Flow chart for tree-branching logic implementation



**Example 27—Software Solution to Logic Function of Figure 20, Using only Byte-Wide Logical Instructions**

```

;BFUNC1 SOLVE A RANDOM LOGIC FUNCTION OF 6
; VARIABLES BY LOADING AND MASKING THE APPROPRIATE
; BITS IN THE ACCUMULATOR, THEN EXECUTING CONDITIONAL
; JUMPS BASED ON ZERO CONDITION
; (APPROACH USED BY BYTE-ORIENTED ARCHITECTURES )
; BYTE AND MASK VALUES CORRESPOND TO RESPECTIVE
; BYTE ADDRESS AND BIT POSITION
;
OUTBUF EQU 22H ; OUTPUT PIN STATE MAP
TESTV MOV A,P2
ANL A,#00000100B
JNZ TESTU
MOV A,TCON
ANL A,#00100000B
JZ TESTX
MOV A,P1
ANL A,#00000010B
JNZ TESTX
MOV A,TCON
ANL A,#00000100B
JZ TESTZ
MOV A,20H
ANL A,#00000001B
JZ SETQ
MOV A,21H
ANL A,#00000001B
JZ SETQ
CLRQ MOV A,OUTBUF
ANL A,#11110111B
OUTQ MOV A,OUTBUF
ORL A,#00001000B
MOV OUTBUF,A
MOV P3,A

```

Cumbersome, to say the least, and error prone. It would be hard to prove the above example worked in all cases without an exhaustive test.

Each move/mask/conditional jump instruction sequence may be replaced by a single bit-test instruction thanks to direct bit addressing. But the algorithm would be equally convoluted.

**Example 28—Software Solution to Logic Function of Figure 20, Using only Bit-Test Instructions**

```

;BFUNC2 SOLVE A RANDOM LOGIC FUNCTION OF 6
; VARIABLES BY DIRECTLY POLLING EACH BIT
; (APPROACH USING MCS-51 UNIQUE BIT-TEST
; INSTRUCTION CAPABILITY )
; SYMBOLS USED IN LOGIC DIAGRAM ASSIGNED TO
; CORRESPONDING 8051 BIT ADDRESSES
;
U BIT P1.1
V BIT P2.2
W BIT TFO
X BIT IE1
Y BIT 20H.0
Z BIT 21H.1
Q BIT P3.3
;
TEST_V JB V,TEST_U
JNB W,TEST_X
TEST_U JB U,TEST_Z
TEST_X JB X,TEST_Z
JNB Y,TEST_Z
TEST_Z JB Z,SET_Q
CLR_Q CLR Q
JMP NXTTST
SET_Q SETB Q
NXTTST
; (CONTINUATION OF PROGRAM)

```

A more elegant and efficient 8051 implementation uses the Boolean ANL and ORL functions to generate the output function using straight-line code. These instructions perform the corresponding logical operations between the carry flag ("Boolean Accumulator") and the addressed bit, leaving the result in the carry. Alternate forms of each instruction (specified in the assembly language by placing a slash before the bit name) use the complement of the bit's state as the input operand.

These instructions may be "strung together" to simulate a multiple input logic gate. When finished, the carry flag contains the result, which may be moved directly to the destination or output pin. No flow chart is needed—it is simple to code directly from the logic diagrams in Figure 20.

**Example 29—Software Solution to Logic Function of Figure 20, Using the MCS-51 (TM) Unique Logical Instructions on Boolean Variables**

```

;BFUNC3 SOLVE A RANDOM LOGIC FUNCTION OF 6
; VARIABLES USING STRAIGHT-LINE LOGICAL INSTRUCTIONS
; ON MCS-51 BOOLEAN VARIABLES
;
MOV C,V
ORL C,W ; OUTPUT OF OR GATE
ANL C,U ; OUTPUT OF TOP AND GATE
MOV FO,C ; SAVE INTERMEDIATE STATE
MOV C,X
ANL C,Y ; OUTPUT OF BOTTOM AND GATE
ORL C,FO ; INCLUDE VALUE SAVED ABOVE
ORL C,Z ; INCLUDE LAST INPUT VARIABLE
MOV G,C ; OUTPUT COMPUTED RESULT

```

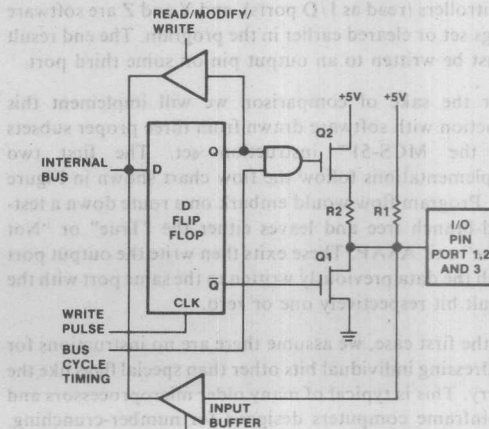
Simplicity itself. Fast, flexible, reliable, easy to design, and easy to debug.

The Boolean features are useful and unique enough to warrant a complete Application Note of their own. Additional uses and ideas are presented in Application Note AP-70, **Using the Intel® MCS-51® Boolean Processing Capabilities**, publication number 121519.

## 5. ON-CHIP PERIPHERAL FUNCTION OPERATION AND INTERFACING

### I/O Ports

The I/O port versatility results from the "quasi-bidirectional" output structure depicted in Figure 22. (This is effectively the structure of ports 1, 2, and 3 for normal I/O operations. On port 0 resistor R2 is disabled except during multiplexed bus operations, providing



**Figure 22. Pseudo-bidirectional I/O port circuitry**

AFN-01502A-30

essentially open-collector outputs. For full electrical characteristics see the User's Manual.)

An output latch bit associated with each pin is updated by direct addressing instructions when that port is the destination. The latch state is buffered to the outside world by R1 and Q1, which may drive a standard TTL input. (In TTL terms, Q1 and R1 resemble an open-collector output with a pull-up resistor to Vcc.)

R2 and Q2 represent an "active pull-up" device enabled momentarily when a 0 previously output changes to a 1. This "jerks" the output pin to a 1 level more quickly than the passive pull-up, improving rise-time significantly if the pin is driving a capacitive load. Note that the active pull-up is **only** activated on 0-to-1 transitions at the output latch (unlike the 8048, in which Q2 is activated whenever a 1 is written out).

Operations using an input port or pin as the source operand use the logic level of the pin itself, rather than the output latch contents. This level is affected by both the microcomputer itself and whatever device the pin is connected to externally. The value read is essentially the "OR-tied" function of Q1 and the external device. If the external device is high-impedance, such as a logic gate input or a three state output in the third state, then reading a pin will reflect the logic level previously output. To use a pin for input, the corresponding output latch must be set. The external device may then drive the pin with either a high or low logic signal. Thus the same port may be used as both input and output by writing ones to all pins used as inputs on output operations, and ignoring all pins used as output on an input operation.

In one operand instructions (INC, DEC, DJNZ and the Boolean CPL) the output latch rather than the input pin level is used as the source data. Similarly, two operand instructions using the port as both one source and the destination (ANL, ORL, XRL) use the output latches. This ensures that latch bits corresponding to pins used as inputs will not be cleared in the process of executing these instructions.

The Boolean operation JBC tests the output latch bit, rather than the input pin, in deciding whether or not to jump. Like the byte-wise logical operations, Boolean operations which modify individual pins of a port leave the other bits of the output latch unchanged.

A good example of how these modes may play together may be taken from the host-processor interface expected by an 8243 I/O expander. Even though the 8051 does not include 8048-type instructions for interfacing with an 8243, the parts can be interconnected (Figure 23) and the protocol may be emulated with simple software.

### Example 30 — Mixing Parallel Output, Input, and Control Strobes on Port 2.

```
IN8243 INPUT DATA FROM AN 8243 I/O EXPANDER
CONNECTED TO P23-P20
P25 & P24 MIMIC CS & A PROG
P27-P26 USED AS INPUTS
PORT TO BE READ IN ACC

IN8243 ORL A, #11010000B
MOV P2, A OUTPUT INSTRUCTION CODE
CLR P2, 4 FALLING EDGE OF PROG
ORL P2, #00001111B SET FOR INPUT
MOV A, P2 READ INPUT DATA
SETB P2, 4 RETURN PROG HIGH
SETB P2, 500 DE-SELECT CHIP
```

### Serial Port and Timer applications

Configuring the 8051's Serial Port for a given data rate and protocol requires essentially three short sections of software. On power-up or hardware reset the serial port and timer control words must be initialized to the appropriate values. Additional software is also needed in the transmit routine to load the serial port data register and in the receive routine to unload the data as it arrives.

This is best illustrated through an arbitrary example. Assume the 8051 will communicate with a CRT operating at 2400 baud (bits per second). Each character is transmitted as seven data bits, odd parity, and one stop bit. This results in a character rate of 2400/10=240 characters per second.

For the sake of clarity, the transmit and receive subroutines are driven by simple-minded software status polling code rather than interrupts. (It might help to refer back to Figures 7-9 showing the control word formats.) The serial port must be initialized to 8-bit UART mode (M0, M1=01), enabled to receive all messages (M2=0, REN=1). The flag indicating that the transmit register is free for more data will be artificially set in order to let the output software know the output register is available. This can all be set up with one instruction.

### Example 31 — Serial Port Mode and Control Bits

```
SPINIT: INITIALIZE SERIAL PORT
FOR 8-BIT UART MODE
& SET TRANSMIT READY FLAG

SPINIT: MOV SCON, #01010010B
```

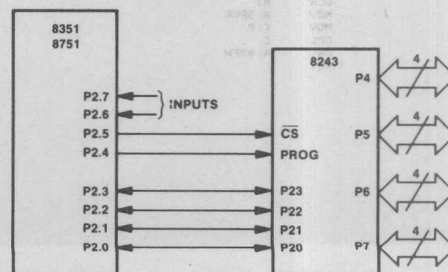


Figure 23. Connecting an 8051 with an 8243 I/O Expander

Timer 1 will be used in auto-reload mode as a data rate generator. To achieve a data rate of 2400 baud, the timer must divide the 1 MHz internal clock by 32 x (desired data rate):

$$\frac{1 \times 10^6}{(32)(2400)}$$

which equals 13.02 rounded down to 13 instruction cycles. The timer must reload the value -13, or 0F3H. (ASM51 will accept both the signed decimal or hexadecimal representations.)

#### Example 32—Initializing Timer Mode and Control Bits

```

T1INIT: INITIALIZE TIMER 1 FOR
        AUTO-RELOAD AT 32*2400 HZ
        (TO USED AS GATED 16-BIT COUNTER)

T1INIT: MOV     TCON, #11010010B
        MOV     TH1, #-13
        SETB    TR1

```

A simple subroutine to transmit the character passed to it in the accumulator must first compute the parity bit, insert it into the data byte, wait until the transmitter is available, output the character, and return. This is nearly as easy said as done.

#### Example 33—Code for UART Output, Adding Parity, Transmitter Loading

```

;SP_OUT ADD ODD PARITY TO ACC AND
;        TRANSMIT WHEN SERIAL PORT READY

SP_OUT: MOV     C, P
        CPL     C
        MOV     ACC, 7-C
        JNB     TI, $
        CLR     TI
        MOV     SBUF, A
        RET

```

A simple minded routine to wait until a character is received, set the carry flag if there is an odd-parity error, and return the masked seven-bit code in the accumulator is equally short.

#### Example 34—Code for UART Reception and Parity Verification

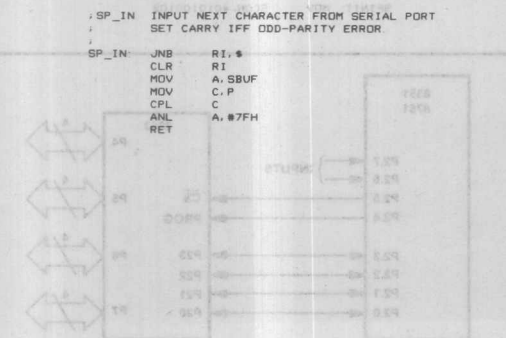


Figure 33. Connecting an 8081 with an 8255 PPI

## 6. SUMMARY

This Application Note has described the architecture, instruction set, and on-chip peripheral features of the first three members of the MCS-51™ microcomputer family. The examples used throughout were admittedly (and necessarily) very simple. Additional examples and techniques may be found in the MCS-51™ User's Manual and other application notes written for the MCS-48™ and MCS-51™ families.

Since its introduction in 1977, the MCS-48™ family has become the industry standard single-chip microcomputer. The MCS-51™ architecture expands the addressing capabilities and instruction set of its predecessor while ensuring flexibility for the future, and maintaining basic software compatibility with the past.

Designers already familiar with the 8048 or 8049 will be able to take with them the education and experience gained from past designs as ever-increasing system performance demands force them to move on to state-of-the-art products. Newcomers will find the power and regularity of the 8051 instruction set an advantage in streamlining both the learning and design processes.

Microcomputer system designers will appreciate the 8051 as basically a single-chip solution to many problems which previously required board-level computers. Designers of real-time control systems will find the high execution speed, on-chip peripherals, and interrupt capabilities vital in meeting the timing constraints of products previously requiring discrete logic designs. And designers of industrial controllers will be able to convert ladder diagrams directly from tested-and-true TTL or relay-logic designs to microcomputer software, thanks to the unique Boolean processing capabilities.

It has not been the intent of this note to gloss over the difficulty of designing microcomputer-based systems. To be sure, the hardware and software design aspects of any new computer system are nontrivial tasks. However, the system speed and level of integration of the MCS-51™ microcomputers, the power and flexibility of the instruction set, and the sophisticated assembler and other support products combine to give both the hardware and software designer as much of a head start on the problem as possible.

A good example of how these modes may play together may be taken from the host-processor interface expected by an 8243 I/O expander. Even though the 8051 does not include 8048-type instructions for interfacing with an 8243, the parts can be interconnected (Figure 33) and the protocol may be emulated with simple software.

## Using the Intel MCS®-51 Boolean Processing Capabilities

## Contents

<b>1. INTRODUCTION</b>	10-32
<b>2. BOOLEAN PROCESSOR OPERATION</b>	10-32
Processing Elements	10-33
Direct Bit Addressing	10-34
Instruction Set	10-39
Simple Instruction Combinations	10-40
<b>3. BOOLEAN PROCESSOR APPLICATIONS</b>	10-41
Design Example #1 — Bit Permutation	10-42
Design Example #2 — Software Serial I/O	10-45
Design Example #3 — Combinatorial Logic Equations	10-46
Design Example #4 — Automotive Dashboard Functions	10-49
Design Example #5 — Complex Control Functions	10-54
Additional Functions and Uses	10-59
<b>4. SUMMARY</b>	10-60
<b>APPENDIX A</b>	10-61



## 1. INTRODUCTION

The Intel microcontroller family now has three new members—the Intel® 8031, 8051, and 8751 single-chip microcomputers. These devices, shown in Figure 1, will allow whole new classes of products to benefit from recent advances in Integrated Electronics. Thanks to Intel's new HMOS® technology, they provide larger program and data memory spaces, more flexible I/O and peripheral capabilities, greater speed, and lower system cost than any previous-generation single-chip microcomputer.

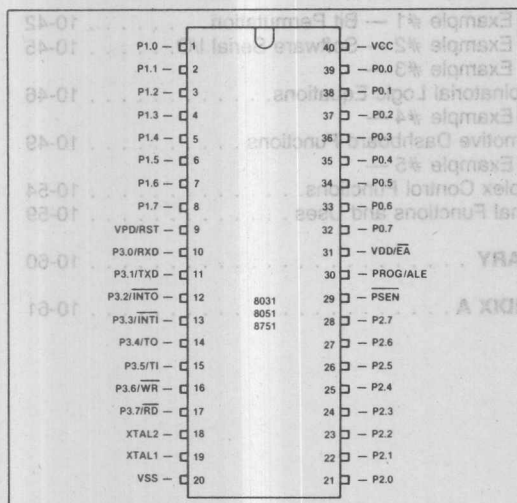


Figure 1. 8051 Family Pinout Diagram.

Table 1 summarizes the quantitative differences between the members of the MCS-48™ and 8051 families. The 8751 contains 4K bytes of EPROM program memory fabricated on-chip, while the 8051 replaces the EPROM with 4K bytes of lower-cost mask-programmed ROM. The 8031 has no program memory on-chip; instead, it accesses up to 64K bytes of program memory from external memory. Otherwise, the three new family members are identical. Throughout this Note, the term "8051" will represent all members of the 8051 Family, unless specifically stated otherwise.

Table 1. Features of Intel's Single-chip Microcomputers.

EPROM Program Memory	ROM Program Memory	External Program Memory	Program Memory (Int/Max)	Data Memory (Bytes)	Instr. Cycle Time	Input/Output Pins	Interrupt Sources	Reg. Banks
—	8021	—	1K 1K	64	10 $\mu$ Sec	21	0	1
—	8022	—	2K 2K	64	10 $\mu$ Sec	28	2	1
8748	8048	8035	1K 4K	64	2.5 $\mu$ Sec	27	2	2
—	8049	8039	2K 4K	128	1.36 $\mu$ Sec	27	2	2
8751	8051	8031	4K 64K	128	1.0 $\mu$ Sec	32	5	4

The CPU in each microcomputer is one of the industry's fastest and most efficient for numerical calculations on byte operands. But controllers often deal with bits, not bytes: in the real world, switch contacts can only be open or closed, indicators should be either lit or dark, motors are either turned on or off, and so forth. For such control situations the most significant aspect of the MCS-51™ architecture is its complete hardware support for one-bit, or *Boolean* variables (named in honor of Mathematician George Boole) as a separate data type.

The 8051 incorporates a number of special features which support the direct manipulation and testing of individual bits and allow the use of single-bit variables in performing logical operations. Taken together, these features are referred to as the MCS-51™ *Boolean Processor*. While the bit-processing capabilities alone would be adequate to solve many control applications, their true power comes when they are used in conjunction with the microcomputer's byte-processing and numerical capabilities.

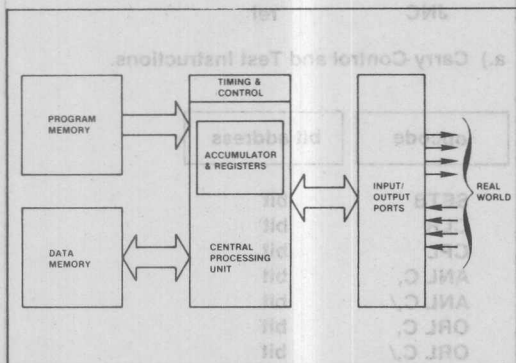
Many concepts embodied by the Boolean Processor will certainly be new even to experienced microcomputer system designers. The purpose of this Application Note is to explain these concepts and show how they are used. It is assumed the reader has read Application Note AP-69, **An Introduction to the Intel® MCS-51™ Single-Chip Microcomputer Family**, publication number 121518, or has been exposed to Intel's single-chip microcomputer product lines.

For detailed information on these parts refer to the **Intel MCS-51™ Family User's Manual**, publication number 121517. The instruction set, assembly language, and use of the 8051 assembler (ASM51) are further described in the **MCS-51™ Macro Assembler User's Guide**, publication number 9800937.

## 2. BOOLEAN PROCESSOR OPERATION

The Boolean Processing capabilities of the 8051 are based on concepts which have been around for some time. Digital computer systems of widely varying designs all have four functional elements in common (Figure 2):

- a central processor (CPU) with the control, timing, and logic circuits needed to execute stored instructions;
- a memory to store the sequence of instructions making up a program or algorithm;
- data memory to store variables used by the program; and
- some means of communicating with the outside world.



**Figure 2. Block Diagram for Abstract Digital Computer.**

The CPU usually includes one or more accumulators or special registers for computing or storing values during program execution. The instruction set of such a processor generally includes, at a minimum, operation classes to perform arithmetic or logical functions on program variables, move variables from one place to another, cause program execution to jump or conditionally branch based on register or variable states, and instructions to call and return from subroutines. The program and data memory functions sometimes share a single memory space, but this is not always the case. When the address spaces are separated, program and data memory need not even have the same basic word width.

A digital computer's flexibility comes in part from combining simple fast operations to produce more complex (albeit slower) ones, which in turn link together eventually solving the problem at hand. A four-bit CPU executing multiple precision subroutines can, for example, perform 64-bit addition and subtraction. The subroutines could in turn be building blocks for floating-point multiplication and division routines. Eventually, the four-bit CPU can simulate a far more complex "virtual" machine.

In fact, *any* digital computer with the above four functional elements can (given time) complete *any* algorithm (though the proverbial room full of chimpanzees at word

processors might first re-create Shakespeare's classics and this Application Note)! This fact offers little consolation to product designers who want programs to run as quickly as possible. By definition, a real-time control algorithm *must* proceed quickly enough to meet the preordained speed constraints of other equipment.

One of the factors determining how long it will take a microcomputer to complete a given chore is the number of instructions it must execute. What makes a given computer architecture particularly well- or poorly-suited for a class of problems is how well its instruction set matches the tasks to be performed. The better the "primitive" operations correspond to the steps taken by the control algorithm, the lower the number of instructions needed, and the quicker the program will run. All else being equal, a CPU supporting 64-bit arithmetic directly could clearly perform floating-point math faster than a machine bogged-down by multiple-precision subroutines. In the same way, direct support for bit manipulation naturally leads to more efficient programs handling the binary input and output conditions inherent in digital control problems.

## Processing Elements

The introduction stated that the 8051's bit-handling capabilities alone would be sufficient to solve some control applications. Let's see how the four basic elements of a digital computer - a CPU with associated registers, program memory, addressable data RAM, and I/O capability - relate to Boolean variables.

**CPU.** The 8051 CPU incorporates special logic devoted to executing several bit-wide operations. All told, there are 17 such instructions, all listed in Table 2. Not shown are 94 other (mostly byte-oriented) 8051 instructions.

**Program Memory.** Bit-processing instructions are fetched from the same program memory as other arithmetic and logical operations. In addition to the instructions of Table 2, several sophisticated program control features like multiple addressing modes, subroutine nesting, and a two-level interrupt structure are useful in structuring Boolean Processor-based programs.

Boolean instructions are one, two, or three bytes long, depending on what function they perform. Those involving only the carry flag have either a single-byte opcode or an opcode followed by a conditional-branch destination byte (Figure 3.a). The more general instructions add a "direct address" byte after the opcode to specify the bit affected, yielding two or three byte encodings (Figure 3.b). Though this format allows potentially 256 directly addressable bit locations, not all of them are implemented in the 8051 family.

**Table 2. MCS-51™ Boolean Processing Instruction Subset.**

Mnemonic	Description	Byte	Cyc
SETB C	Set Carry flag	1	1
SETB bit	Set direct Bit	2	1
CLR C	Clear Carry flag	1	1
CLR bit	Clear direct bit	2	1
CPL C	Complement Carry flag	1	1
CPL bit	Complement direct bit	2	1
MOV C,bit	Move direct bit to Carry flag	2	1
MOV bit,C	Move Carry flag to direct bit	2	2
ANL C,bit	AND direct bit to Carry flag	2	2
ANL C, bit	AND complement of direct bit to Carry flag	2	2
ORL C,bit	OR direct bit to Carry flag	2	2
ORL C, bit	OR complement of direct bit to Carry flag	2	2
JC rel	Jump if Carry is flag is set	2	2
JNC rel	Jump if No Carry flag	2	2
JB bit,rel	Jump if direct Bit set	3	2
JNB bit,rel	Jump if direct Bit Not set	3	2
JBC bit,rel	Jump if direct Bit is set & Clear bit	3	2

**Address mode abbreviations:**

C — Carry flag.

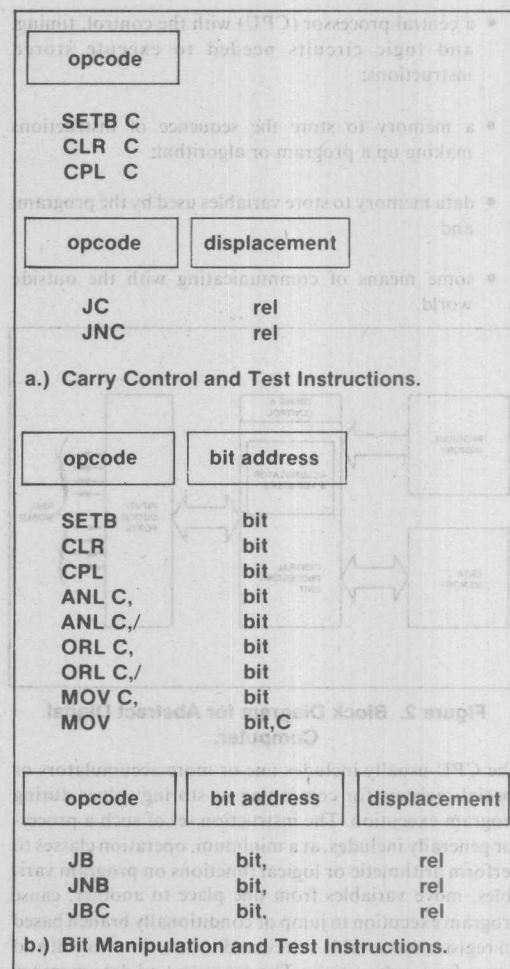
bit — 128 software flags, any I/O pin, control or status bit

rel — All conditional jumps include an 8-bit offset byte. Range is +127 -128 bytes relative to first byte of the following instruction.

All mnemonics copyrighted© Intel Corporation 1980

**Data Memory.** The instructions in Figure 3.b can operate directly upon 144 general purpose bits forming the Boolean processor "RAM." These bits can be used as software flags or to store program variables. Two operand instructions use the CPU's carry flag ("C") as a special one-bit register; in a sense, the carry is a "Boolean accumulator" for logical operations and data transfers.

**Input/Output.** All 32 I/O pins can be addressed as individual inputs, outputs, or both, in any combination. Any pin can be a control strobe output, status (Test) input, or serial I/O link implemented via software. An additional 33 individually addressable bits reconfigure, control, and monitor the status of the CPU and all on-chip peripheral functions (timer counters, serial port modes, interrupt logic, and so forth).



**Figure 3. Bit Addressing Instruction Formats.**

### Direct Bit Addressing

The most significant bit of the direct address byte selects one of two groups of bits. Values between 0 and 127 (00H and 7FH) define bits in a block of 32 bytes of on-chip RAM: between RAM addresses 20H and 2FH (Figure 4.a). They are numbered consecutively from the lowest-order byte's lowest-order bit through the highest-order byte's highest-order bit.

Bit addresses between 128 and 255 (80H and 0FFH) correspond to bits in a number of special registers, mostly used for I/O or peripheral control. These positions are numbered with a different scheme than RAM: the five high-order address bits match those of the register's own address, while the three low-order bits identify the bit position within that register (Figure 4.b).

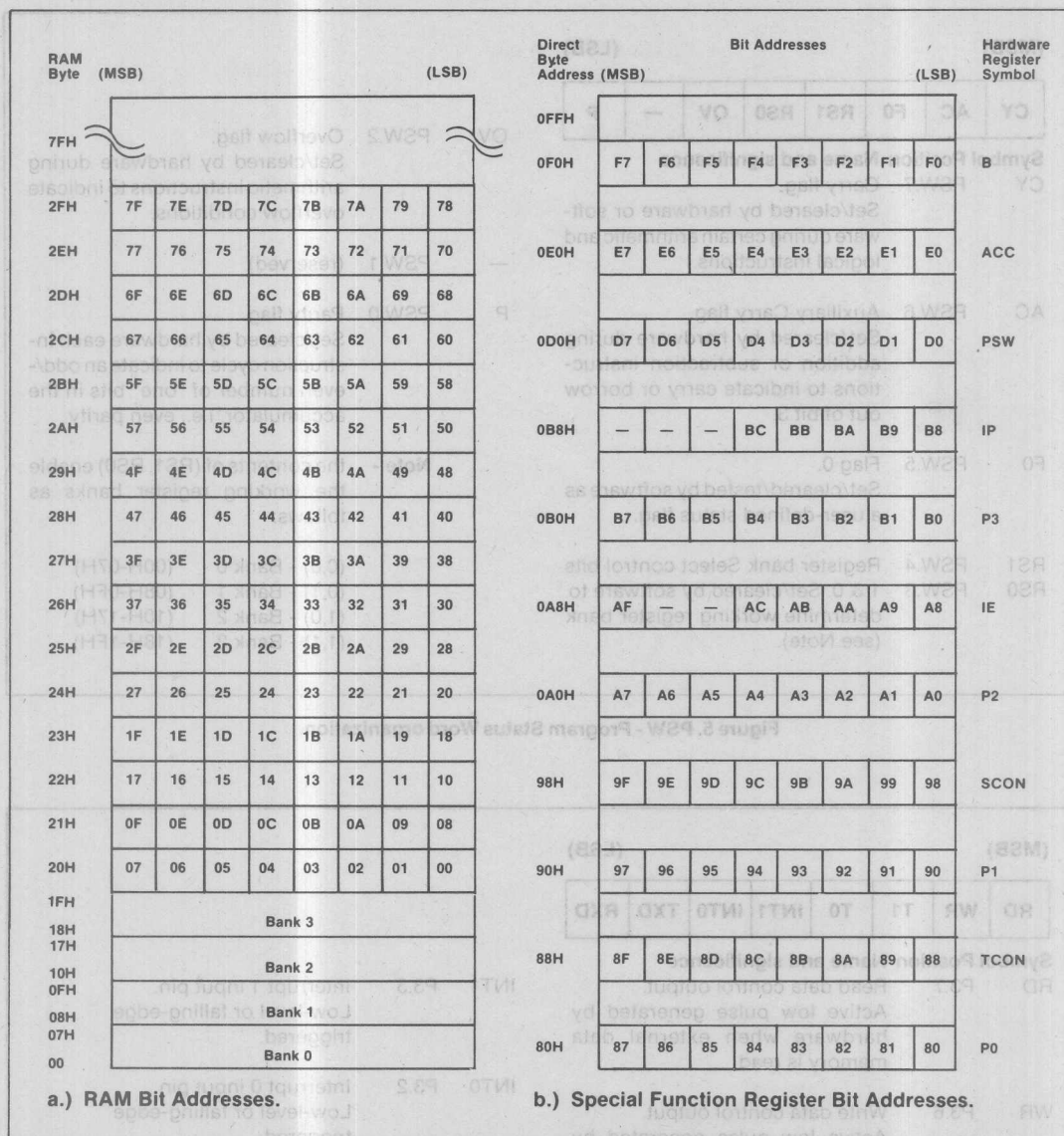


Figure 4. Bit Address Maps.

Notice the column labeled "Symbol" in Figure 5. Bits with special meanings in the PSW and other registers have corresponding symbolic names. General-purpose (as opposed to carry-specific) instructions may access the carry like any other bit by using the mnemonic CY in place of C. P0, P1, P2, and P3 are the 8051's four I/O ports; secondary functions assigned to each of the eight pins of P3 are shown in Figure 6.

Figure 7 shows the last four bit addressable registers. TCON (Timer Control) and SCON (Serial port Control) control and monitor the corresponding peripherals, while IE (Interrupt Enable) and IP (Interrupt Priority) enable and prioritize the five hardware interrupt sources. Like the reserved hardware register addresses, the five bits not implemented in IE and IP should not be accessed; they can *not* be used as software flags.



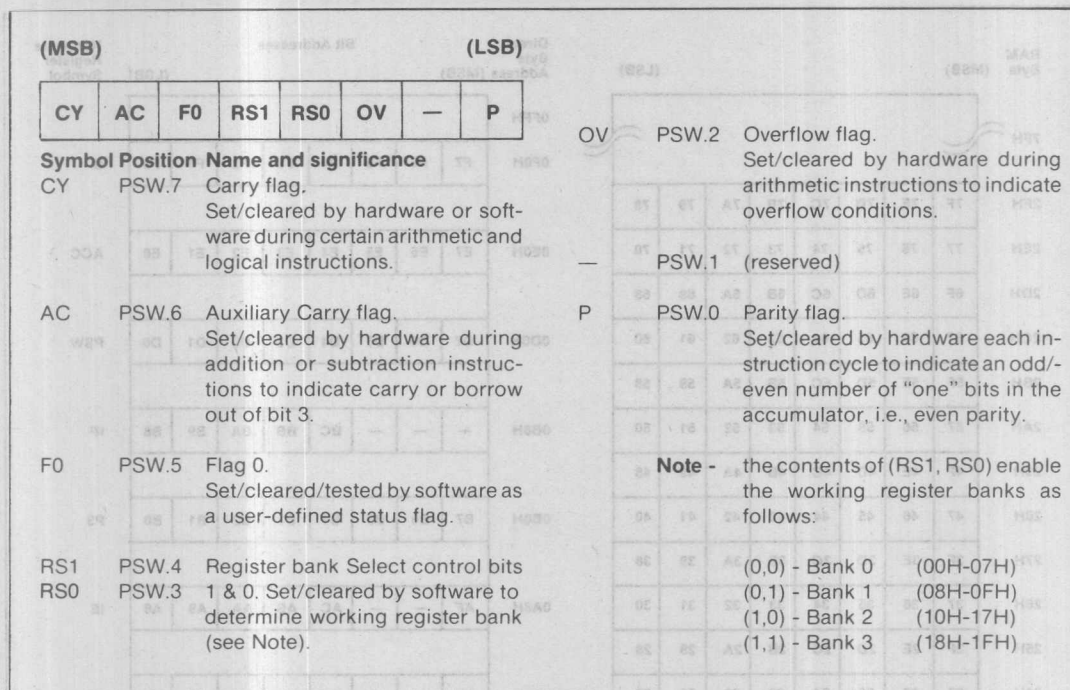


Figure 5. PSW - Program Status Word organization.

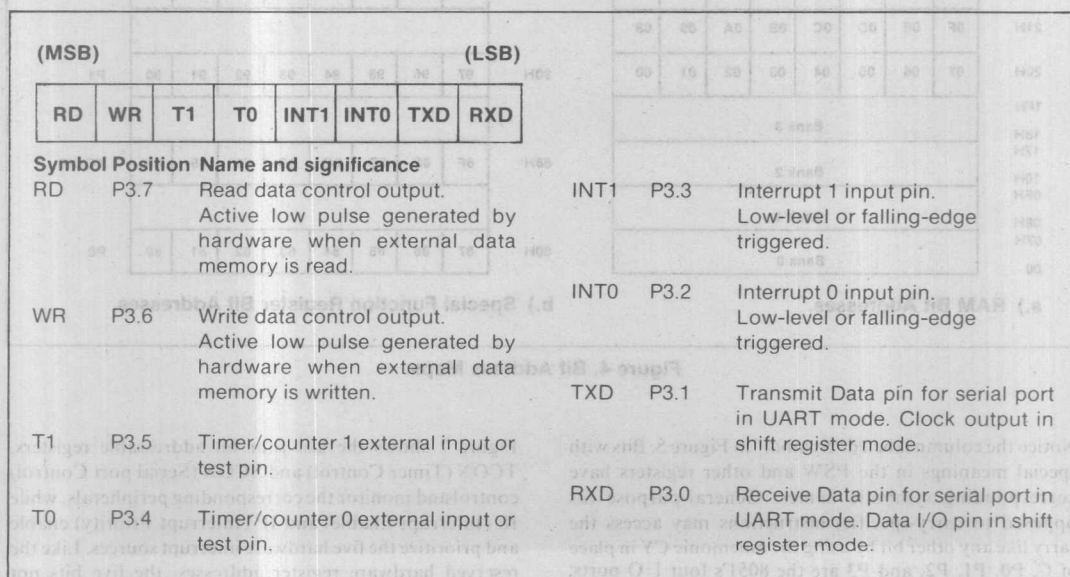


Figure 6. P3 - Alternate I/O Functions of Port 3.

TF1	TR1	TF0	TR0	IE1	IT1	IE0	IT0
-----	-----	-----	-----	-----	-----	-----	-----

**TF1** TCON.7 Timer 1 overflow Flag.  
Set by hardware on timer/counter overflow. Cleared when interrupt processed.

TR1 TCON.6 Timer 1 Run control bit.  
Set/cleared by software to turn  
timer/counter on/off.

**TF0** TCON.5 Timer 0 overflow Flag.  
Set by hardware on  
timer/counter overflow. Cleared  
when interrupt processed.

TR0 TCON.4 Timer 0 Run control bit.  
Set/cleared by software to turn  
timer/counter on/off.

(MSB) (LSB)

SM0	SM1	SM2	REN	TB8	RB8	TI	RI
-----	-----	-----	-----	-----	-----	----	----

SM0 SCON.7 Serial port Mode control bit 0.  
Set/cleared by software (see  
note).

SM1 SCON.6 Serial port Mode control bit 1.  
Set/cleared by software (see note).

**SM2**    **SCON.5** Serial port Mode control bit 2.  
Set by software to disable reception of frames for which bit 8 is zero.

REN    SCON.4 Receiver Enable control bit.  
Set/cleared by software to  
enable/disable serial data  
reception.

<p><b>TB8</b>    <b>SCON.3</b> Transmit Bit 8.</p> <p>Set/cleared by hardware to determine state of ninth data bit transmitted in 9-bit UART mode.</p>	<p><b>SCON.2</b>    <b>SMOD</b> Serial Mode Divider.</p> <p>When SMOD = 1, the baud rate is divided by 2. When SMOD = 0, the baud rate is not divided by 2.</p>
--	---

b.) **SCON** - Serial Port Control/status register.

**IE1** TCON.3 Interrupt 1 Edge flag.  
Set by hardware when external interrupt edge detected. Cleared when interrupt processed.

**IT1** TCON.2 Interrupt 1 Type control bit.  
Set/cleared by software to specify falling edge/low level triggered external interrupts.

**IE0** TCON.1 Interrupt 0 Edge flag.  
Set by hardware when external interrupt edge detected. Cleared when interrupt processed.

**IT0** TCON.0 Interrupt 0 Type control bit.  
Set/cleared by software to specify falling edge/low level triggered external interrupts.

RB8    **SCON.2 Receive Bit 8.**  
Set/cleared by hardware to indicate state of ninth data bit received.

TI     SCON.1 Transmit Interrupt flag.  
Set by hardware when byte  
transmitted. Cleared by software  
after servicing.

RI	SCON.0 Receive Interrupt flag. Set by hardware when byte received. Cleared by software after servicing.
----	--

**Note -** the state of (SM0,SM1) selects:

- (0,0) - Shift register I/O expansion.
- (0,1) - 8 bit UART, variable data rate.
- (1,0) - 9 bit UART, fixed data rate.
- (1,1) - 9 bit UART, variable data rate.

**Figure 7. Peripheral Configuration Registers.**

Symbol	Position	Name and significance
$\alpha$	1	Angle of attack
$\beta$	2	Angle of sideslip
$\gamma$	3	Angle of yaw
$\delta$	4	Angle of roll
$\epsilon$	5	Angle of pitch
$\zeta$	6	Angle of heave
$\eta$	7	Angle of sway
$\theta$	8	Angle of roll
$\phi$	9	Angle of pitch
$\psi$	10	Angle of yaw
$\chi$	11	Angle of roll
$\omega$	12	Angle of pitch
$\nu$	13	Angle of yaw
$\mu$	14	Angle of roll
$\lambda$	15	Angle of pitch
$\kappa$	16	Angle of yaw
$\iota$	17	Angle of roll
$\theta$	18	Angle of pitch
$\phi$	19	Angle of yaw
$\psi$	20	Angle of roll
$\chi$	21	Angle of pitch
$\omega$	22	Angle of yaw
$\nu$	23	Angle of roll
$\mu$	24	Angle of pitch
$\lambda$	25	Angle of yaw
$\kappa$	26	Angle of roll
$\iota$	27	Angle of pitch
$\theta$	28	Angle of yaw
$\phi$	29	Angle of roll
$\psi$	30	Angle of pitch
$\chi$	31	Angle of yaw
$\omega$	32	Angle of roll
$\nu$	33	Angle of pitch
$\mu$	34	Angle of yaw
$\lambda$	35	Angle of roll
$\kappa$	36	Angle of pitch
$\iota$	37	Angle of yaw
$\theta$	38	Angle of roll
$\phi$	39	Angle of pitch
$\psi$	40	Angle of yaw
$\chi$	41	Angle of roll
$\omega$	42	Angle of pitch
$\nu$	43	Angle of yaw
$\mu$	44	Angle of roll
$\lambda$	45	Angle of pitch
$\kappa$	46	Angle of yaw
$\iota$	47	Angle of roll
$\theta$	48	Angle of pitch
$\phi$	49	Angle of yaw
$\psi$	50	Angle of roll
$\chi$	51	Angle of pitch
$\omega$	52	Angle of yaw
$\nu$	53	Angle of roll
$\mu$	54	Angle of pitch
$\lambda$	55	Angle of yaw
$\kappa$	56	Angle of roll
$\iota$	57	Angle of pitch
$\theta$	58	Angle of yaw
$\phi$	59	Angle of roll
$\psi$	60	Angle of pitch
$\chi$	61	Angle of yaw
$\omega$	62	Angle of roll
$\nu$	63	Angle of pitch
$\mu$	64	Angle of yaw
$\lambda$	65	Angle of roll
$\kappa$	66	Angle of pitch
$\iota$	67	Angle of yaw
$\theta$	68	Angle of roll
$\phi$	69	Angle of pitch
$\psi$	70	Angle of yaw
$\chi$	71	Angle of roll
$\omega$	72	Angle of pitch
$\nu$	73	Angle of yaw
$\mu$	74	Angle of roll
$\lambda$	75	Angle of pitch
$\kappa$	76	Angle of yaw
$\iota$	77	Angle of roll
$\theta$	78	Angle of pitch
$\phi$	79	Angle of yaw
$\psi$	80	Angle of roll
$\chi$	81	Angle of pitch
$\omega$	82	Angle of yaw
$\nu$	83	Angle of roll
$\mu$	84	Angle of pitch
$\lambda$	85	Angle of yaw
$\kappa$	86	Angle of roll
$\iota$	87	Angle of pitch
$\theta$	88	Angle of yaw
$\phi$	89	Angle of roll
$\psi$	90	Angle of pitch
$\chi$	91	Angle of yaw
$\omega$	92	Angle of roll
$\nu$	93	Angle of pitch
$\mu$	94	Angle of yaw
$\lambda$	95	Angle of roll
$\kappa$	96	Angle of pitch
$\iota$	97	Angle of yaw
$\theta$	98	Angle of roll
$\phi$	99	Angle of pitch
$\psi$	100	Angle of yaw

- IE.6 (reserved)
- IE.5

ET1 IE3 Enable Timer 1 control bit.  
Set/cleared by software to  
enable/ disable interrupts from  
timer/counter 1.

## (MSB) (LSB)

Symbol	Position	Name and significance
1	1	1
2	2	2
3	3	3
4	4	4
5	5	5
6	6	6
7	7	7
8	8	8
9	9	9
10	10	10
11	11	11
12	12	12
13	13	13
14	14	14
15	15	15
16	16	16
17	17	17
18	18	18
19	19	19
20	20	20
21	21	21
22	22	22
23	23	23
24	24	24
25	25	25
26	26	26
27	27	27
28	28	28
29	29	29
30	30	30
31	31	31
32	32	32
33	33	33
34	34	34
35	35	35
36	36	36
37	37	37
38	38	38
39	39	39
40	40	40
41	41	41
42	42	42
43	43	43
44	44	44
45	45	45
46	46	46
47	47	47
48	48	48
49	49	49
50	50	50
51	51	51
52	52	52
53	53	53
54	54	54
55	55	55
56	56	56
57	57	57
58	58	58
59	59	59
60	60	60
61	61	61
62	62	62
63	63	63
64	64	64
65	65	65
66	66	66
67	67	67
68	68	68
69	69	69
70	70	70
71	71	71
72	72	72
73	73	73
74	74	74
75	75	75
76	76	76
77	77	77
78	78	78
79	79	79
80	80	80
81	81	81
82	82	82
83	83	83
84	84	84
85	85	85
86	86	86
87	87	87
88	88	88
89	89	89
90	90	90
91	91	91
92	92	92
93	93	93
94	94	94
95	95	95
96	96	96
97	97	97
98	98	98
99	99	99
100	100	100

PT1	IP.3	Timer 1 Priority control bit. Set/cleared by software to specify high/low priority interrupts for timer/counter 1.
-----	------	---

170	160	150	140	130	120	110	100	90	80	70	60	50	40	30	20	10	0
-----	-----	-----	-----	-----	-----	-----	-----	----	----	----	----	----	----	----	----	----	---

EX0	IE.0	Enable External interrupt 0 control bit. Set/cleared by software to enable/disable interrupts from INTO.
-----	------	--

**PX0** **IP.0** External interrupt 0 Priority control bit. Set/cleared by software to specify high/low priority interrupts for INT0.

The accumulator and B registers (A and B) are normally involved in byte-wide arithmetic, but their individual bits can also be used as 16 general software flags. Added with the 128 flags in RAM, this gives 144 general purpose variables for bit-intensive programs. The program status word (PSW) in Figure 5 is a collection of flags and machine status bits including the carry flag itself. Byte operations acting on the PSW can therefore affect the carry.

## Instruction Set

Having looked at the bit variables available to the Boolean Processor, we will now look at the four classes of instructions that manipulate these bits. It may be helpful to refer back to Table 2 while reading this section.

**State Control.** Addressable bits or flags may be set, cleared, or logically complemented in one instruction cycle with the two-byte instructions SETB, CLR, and CPL. (The "B" affixed to SETB distinguishes it from the assembler "SET" directive used for symbol definition.) SETB and CLR are analogous to loading a bit with a constant: 1 or 0. Single byte versions perform the same three operations on the carry.

The MCS-51™ assembly language specifies a bit address in any of three ways:

- by a number or expression corresponding to the direct bit address (0-255);
- by the name or address of the register containing the bit, the *dot operator* symbol (a period: "."), and the bit's position in the register (7-0);
- in the case of control and status registers, by the predefined assembler symbols listed in the first columns of Figures 5-7.

Bits may also be given user-defined names with the assembler "BIT" directive and any of the above techniques. For example, bit 5 of the PSW may be cleared by any of the four instructions.

USR_FLG	BIT	PSW.5	: User Symbol Definition
CLR		0D5H	: Absolute Addressing
CLR		PSW.5	: Use of Dot Operator
CLR		F0	: Pre-Defined Assembler Symbol
CLR		USR_FLG	: User-Defined Symbol

**Data Transfers.** The two-byte MOV instructions can transport any addressable bit to the carry in one cycle, or copy the carry to the bit in two cycles. A bit can be moved between two arbitrary locations via the carry by combining the two instructions. (If necessary, push and pop the PSW to preserve the previous contents of the carry.) These instructions can replace the multi-instruction sequence of Figure 8, a program structure appearing in controller applications whenever flags or outputs are conditionally switched on or off.

**Logical Operations.** Four instructions perform the logical-AND and logical-OR operations between the carry and another bit, and leave the results in the carry. The instruction mnemonics are ANL and ORL; the absence or presence of a

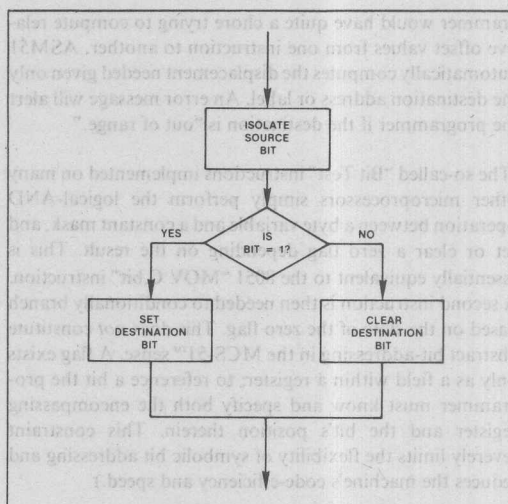


Figure 8. Bit Transfer Instruction Operation.

slash mark ("/) before the source operand indicates whether to use the positive-logic value or the logical complement of the addressed bit. (The source operand itself is never affected.)

**Bit-test Instructions.** The conditional jump instructions "JC rel" (Jump on Carry) and "JNC rel" (Jump on Not Carry) test the state of the carry flag, branching if it is a one or zero, respectively. (The letters "rel" denote relative code addressing.) The three-byte instructions "JB bit, rel" and "JNB bit, rel" (Jump on Bit and Jump on Not Bit) test the state of any addressable bit in a similar manner. A fifth instruction combines the Jump on Bit and Clear operations. "JBC bit, rel" conditionally branches to the indicated address, then clears the bit in the same two cycle instruction. This operation is the same as the MCS-48™ "JTF" instructions.

All 8051 conditional jump instructions use program counter-relative addressing, and all execute in two cycles. The last instruction byte encodes a signed displacement ranging from -128 to +127. During execution, the CPU adds this value to the incremented program counter to produce the jump destination. Put another way, a conditional jump to the immediately following instruction would encode 00H in the offset byte.

A section of program or subroutine written using only relative jumps to nearby addresses will have the same machine code independent of the code's location. An assembled routine may be repositioned anywhere in memory, even crossing memory page boundaries, without having to modify the program or recompute destination addresses. To facilitate this flexibility, there is an unconditional "Short Jump" (SJMP) which uses relative addressing as well. Since a pro-



programmer would have quite a chore trying to compute relative offset values from one instruction to another. ASM51 automatically computes the displacement needed given only the destination address or label. An error message will alert the programmer if the destination is "out of range."

(The so-called "Bit Test" instructions implemented on many other microprocessors simply perform the logical-AND operation between a byte variable and a constant mask, and set or clear a zero flag depending on the result. This is essentially equivalent to the 8051 "MOV C,bit" instruction. A second instruction is then needed to conditionally branch based on the state of the zero flag. This does *not* constitute abstract bit-addressing in the MCS-51™ sense. A flag exists only as a field within a register; to reference a bit the programmer must know and specify both the encompassing register and the bit's position therein. This constraint severely limits the flexibility of symbolic bit addressing and reduces the machine's code-efficiency and speed.)

**Interaction with Other Instructions.** The carry flag is also affected by the instructions listed in Table 3. It can be rotated through the accumulator, and altered as a side effect of arithmetic instructions. Refer to the User's Manual for details on how these instructions operate.

### Simple Instruction Combinations

By combining general purpose bit operations with certain addressable bits, one can "custom build" several hundred useful instructions. All eight bits of the PSW can be tested directly with conditional jump instructions to monitor (among other things) parity and overflow status. Programmers can take advantage of 128 software flags to keep track of operating modes, resource usage, and so forth.

The Boolean instructions are also the most efficient way to control or reconfigure peripheral and I/O registers. All 32 I/O lines become "test pins," for example, tested by conditional jump instructions. Any output pin can be toggled (complemented) in a single instruction cycle. Setting or clearing the Timer Run flags (TR0 and TR1) turn the timer/counters on or off; polling the same flags elsewhere lets the program determine if a timer is running. The respective overflow flags (TF0 and TF1) can be tested to determine when the desired period or count has elapsed, then cleared in preparation for the next repetition. (For the record, these bits are all part of the TCON register, Figure 7.a. Thanks to symbolic bit addressing, the programmer only needs to remember the mnemonic associated with each function. In other words, don't bother memorizing control word layouts.)

In the MCS-48® family, instructions corresponding to some of the above functions require specific opcodes. Ten different opcodes serve to clear complement the software flags F0 and F1, enable/disable each interrupt, and start/stop the timer. In the 8051 instruction set, just three opcodes (SETB,

**Table 3. Other Instructions Affecting the Carry Flag.**

Mnemonic	Description	Byte	Cyc
ADD A,Rn	Add register to Accumulator	1	1
ADD A,direct	Add direct byte to Accumulator	2	1
ADD A,@Ri	Add indirect RAM to Accumulator	1	1
ADD A,#data	Add immediate data to Accumulator	2	1
ADDC A,Rn	Add register to Accumulator with Carry flag	1	1
ADDC A,direct	Add direct byte to Accumulator with Carry flag	2	1
ADDC A,@Ri	Add indirect RAM to Accumulator with Carry flag	1	1
ADDC A,#data	Add immediate data to Acc with Carry flag	2	1
SUBB A,Rn	Subtract register from Accumulator with borrow	1	1
SUBB A,direct	Subtract direct byte from Acc with borrow	2	1
SUBB A,@Ri	Subtract indirect RAM from Acc with borrow	1	1
SUBB A,#data	Subtract immediate data from Acc with borrow	2	1
MUL AB	Multiply A & B	1	4
DIV AB	Divide A by B	1	4
DA A	Decimal Adjust Accumulator	1	1
RLC A	Rotate Accumulator Left through the Carry flag	1	1
RRC A	Rotate Accumulator Right through Carry flag	1	1
CJNE A,direct,rel	Compare direct byte to Acc & Jump if Not Equal	3	2
CJNE A,#data,rel	Compare immediate to Acc & Jump if Not Equal	3	2
CJNE Rn,#data,rel	Compare immed to register & Jump if Not Equal	3	2
CJNE @Ri,#data,rel	Compare immed to indirect & Jump if Not Equal	3	2

All mnemonics copyrighted © Intel Corporation 1980.

CLR, CPL) with a direct bit address appended perform the same functions. Two test instructions (JB and JNB) can be combined with bit addresses to test the software flags, the 8048 I/O pins T0, T1, and INT, and the eight accumulator bits, replacing 15 more different instructions.

Table 4.a shows how 8051 programs implement software flag and machine control functions associated with special

using awkward sequences of other basic operations. As mentioned earlier, any CPU can solve any problem given enough time.

*Quantitatively*, the differences between a solution allowed by the 8051 and those required by previous architectures are numerous. What the 8051 Family buys you is a faster, cleaner, lower-cost solution to microcontroller applications.

The opcode space freed by condensing many specific 8048

**Table 4.a. Contrasting 8048 and 8051 Bit Control and Testing Instructions.**

8048 Instruction	Bytes	Cycles	uSec	8x51 Instruction	Bytes	Cycles & uSec
Flag Control						
CLR C	1	1	2.5	CLR C	1	1
CPL F0	1	1	2.5	CPL F0	2	1
Flag Testing						
JNC offset	2	2	5.0	JNC rel	2	2
JF0 offset	2	2	5.0	JB F0,rel	3	2
JB7 offset	2	2	5.0	JB ACC.7,rel	3	2
Peripheral Polling						
JT0 offset	2	2	5.0	JB T0,rel	3	2
JN1 offset	2	2	5.0	JNB INT0,rel	3	2
JTF offset	2	2	5.0	JBC TF0,rel	3	2
Machine and Peripheral Control						
STRT T	1	1	2.5	SETB TR0	2	1
EN I	1	1	2.5	SETB EX0	2	1
DIS TCNT1	1	1	2.5	CLR ET0	2	1

**Table 4.b. Replacing 8048 instruction sequences with single 8x51 instructions.**

8048 Instructions	Bytes	Cycles	uSec	8051 Instructions	Bytes	Cycles & uSec
Flag Control						
Set carry:						
CLR C	1	1	2.5	SETB C	1	1
CPL C	1	1	2.5			
Set Software Flag:						
CLR F0	1	1	2.5	SETB F0	2	1
CPL F0	1	1	2.5			

opcodes in the 8048. In every case the MCS-51™ solution requires the same number of machine cycles, and executes 2.5 times faster.

### 3. BOOLEAN PROCESSOR APPLICATIONS

So what? Then what does all this buy you?

*Qualitatively*, nothing. All the same capabilities could be (and often have been) implemented on other machines

instructions into a few general operations has been used to add new functionality to the MCS-51™ architecture - both for byte and bit operations. 144 software flags replace the 8048's two. These flags (and the carry) may be directly set, not just cleared and complemented, and all can be tested for either state, not just one. Operating mode bits previously inaccessible may be read, tested, or saved. Situations where the 8051 instruction set provides new capabilities are contrasted with 8048 instruction sequences in Table 4.b. Here the 8051 speed advantage ranges from 5x to 15x!

Table 4b. (Continued)

8048 Instructions	Bytes	Cycles	uSec	8x51 Instructions	Bytes	Cycles & uSec
Turn Off Output Pin: ANL P1.#0FBH =	2	2	5.0	CLR P1.2	2	1
Complement Output Pin: IN A.PI XRL A.#04H OUTL P1.A =	4	6	15.0	CPL P1.2	2	1
Clear Flag in RAM: MOV R0.#FLGADR MOV A.@R0 ANL A.#FLGMASK MOV @R0.A =	6	6	15.0	CLR USER_FLG	2	
Flag Testing Jump if Software Flag is 0: JF0 \$+4 JMP offset =	4	4	10.0	JNB F0,rel	3	2
Jump if Accumulator bit is 0: CPL A JB7 offset CPL A =	4	4	10.0	JNB ACC.7,rel	3	2
Peripheral Polling Test if Input Pin is Grounded: IN A.PI CPL A JB3 offset =	4	5	12.5	JNB P1.3,rel	3	2
Test if Interrupt Pin is High: JNI \$+4 JMP offset =	4	4	10.0	JB INT0,rel	3	2

Combining Boolean and byte-wide instructions can produce great synergy. An MCS-51™ based application will prove to be:

- simpler to write since the architecture correlates more closely with the problems being solved;
- easier to debug because more individual instructions have no unexpected or undesirable side-effects;
- more byte efficient due to direct bit addressing and program counter relative branching;
- faster running because fewer bytes of instruction need to be fetched and fewer conditional jumps are processed;
- lower cost because of the high level of system-intergration within one component.

These rather unabashed claims of excellence shall not go unsubstantiated. The rest of this chapter examines less trivial tasks simplified by the Boolean processor. The first

three compare the 8051 with other microprocessors; the last two go into 8051-based system designs in much greater depth.

### Design Example #1 - Bit Permutation

First off, we'll use the bit-transfer instructions to permute a lengthy pattern of bits.

A steadily increasing number of data communication products use encoding methods to protect the security of sensitive information. By law, interstate financial transactions involving the Federal banking system must be transmitted using the Federal Information Processing *Data Encryption Standard* (DES).

Basically, the DES combines eight bytes of "plaintext" data (in binary, ASCII, or any other format) with a 56-bit "key", producing a 64-bit encrypted value for transmission. At the receiving end the same algorithm is applied to the incoming data using the same key, reproducing the original eight byte message. The algorithm used for these permutations is fixed; different user-defined keys ensure data privacy.

It is not the purpose of this note to describe the DES in any detail. Suffice it to say that encryption/decryption is a long, iterative process consisting of rotations, exclusive-OR operations, function table look-ups, and an extensive (and quite bizarre) sequence of bit permutation, packing, and unpacking steps. (For further details refer to the June 21, 1979 issue of **Electronics** magazine.) The bit manipulation steps are included, it is rumored, to impede a general purpose digital supercomputer trying to "break" the code. Any algorithm implementing the DES with previous generation microprocessors would spend virtually all of its time diddling bits.

The bit manipulation performed is typified by the Key Schedule Calculation represented in Figure 9. This step is repeated 16 times for each key used in the course of a transmission. In essence, a seven-byte, 56-bit "Shifted Key Buffer" is transformed into an eight-byte, "Permutation Buffer" without altering the shifted Key. The arrows in Figure 9 indicate a few of the translation steps. Only six bits of each byte of the Permutation Buffer are used; the two high-order bits of each byte are cleared. This means only 48 of the 56 Shifted Key Buffer bits are used in any one iteration.

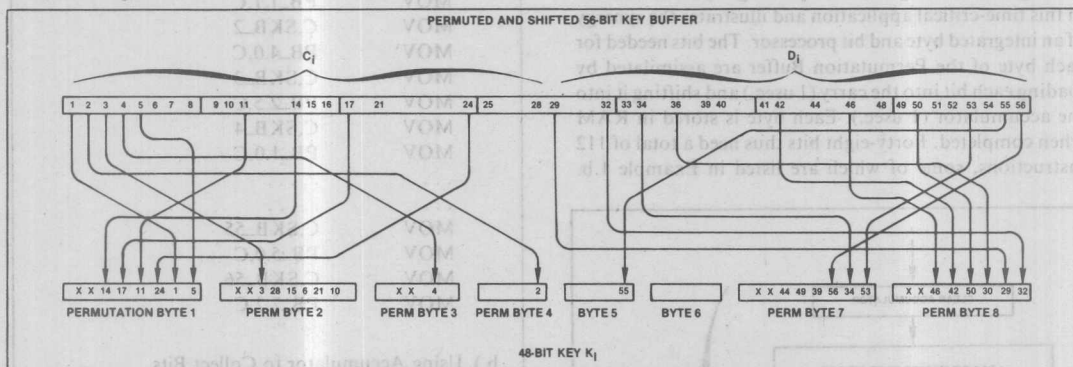


Figure 9. DES Key Schedule Transformation.

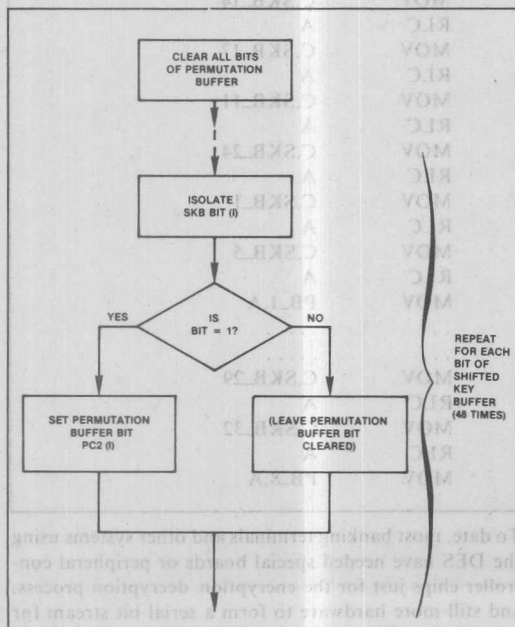


Figure 10.a. Flowchart for Key permutation attempted with a byte processor.

Different microprocessor architectures would best implement this type of permutation in different ways. Most approaches would share the steps of Figure 10.a:

- Initialize the Permutation Buffer to default state (ones or zeroes);
- Isolate the state of a bit of a byte from the Key Buffer. Depending on the CPU, this might be accomplished by rotating a word of the Key Buffer through a carry flag or testing a bit in memory or an accumulator against a mask byte;
- Perform a conditional jump based on the carry or zero flag if the Permutation Buffer default state is correct;
- Otherwise reverse the corresponding bit in the permutation buffer with logical operations and mask bytes.

Each step above may require several instructions. The last three steps must be repeated for all 48 bits. Most microprocessors would spend 300 to 3,000 microseconds on each of the 16 iterations.

Notice, though, that this flow chart looks a lot like Figure 8. The Boolean Processor can permute bits by simply moving



them from the source to the carry to the destination—a total of two instructions taking four bytes and three microseconds per bit. Assume the Shifted Key Buffer and Permutation Buffer both reside in bit-addressable RAM, with the bits of the former assigned symbolic names SKB\_1, SKB\_2, ..., SKB\_56, and that the bytes of the latter are named PB\_1, ..., PB\_8. Then working from Figure 9, the software for the permutation algorithm would be that of Example 1.a. The total routine length would be 192 bytes, requiring 144 microseconds.

The algorithm of Figure 10.b is just slightly more efficient in this time-critical application and illustrates the synergy of an integrated byte and bit processor. The bits needed for each byte of the Permutation Buffer are assimilated by loading each bit into the carry (1 usec.) and shifting it into the accumulator (1 usec.). Each byte is stored in RAM when completed. Forty-eight bits thus need a total of 112 instructions, some of which are listed in Example 1.b.

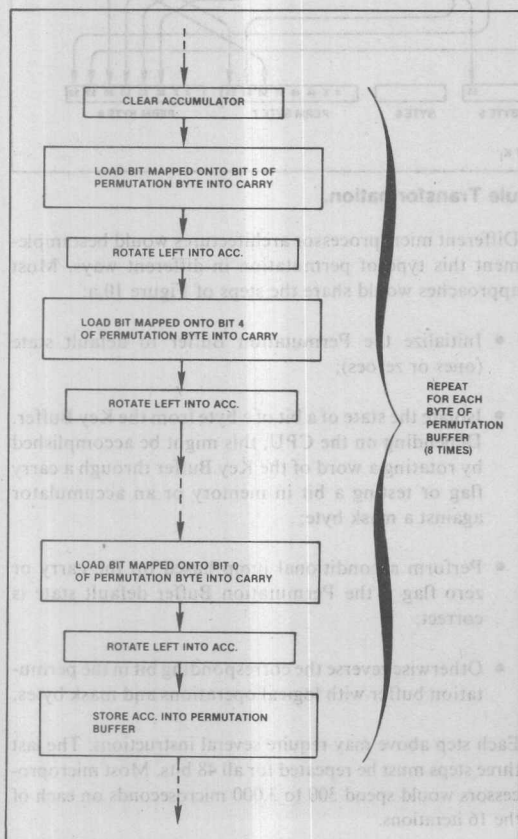


Figure 10.b. DES Key Permutation with Boolean Processor.

Worst-case execution time would be 112 microseconds, since each instruction takes a single cycle. Routine length would also decrease, to 168 bytes. (Actually, in the context of the complete encryption algorithm, each permuted byte would be processed as soon as it is assimilated—saving memory and cutting execution time by another 8 usec.)

#### Example 1. DES Key Permutation Software.

##### a.) "Brute Force" technique.

```

MOV    C.SKB_1
MOV    PB_1.1.C
MOV    C.SKB_2
MOV    PB_4.0.C
MOV    C.SKB_3
MOV    PB_2.5.C
MOV    C.SKB_4
MOV    PB_1.0.C
...
MOV    C.SKB_55
MOV    PB_5.0.C
MOV    C.SKB_56
MOV    PB_7.2.C
  
```

##### b.) Using Accumulator to Collect Bits.

```

CLR    A
MOV    C.SKB_14
RLC    A
MOV    C.SKB_17
RLC    A
MOV    C.SKB_11
RLC    A
MOV    C.SKB_24
RLC    A
MOV    C.SKB_1
RLC    A
MOV    C.SKB_5
RLC    A
MOV    PB_1.A
...
MOV    C.SKB_29
RLC    A
MOV    C.SKB_32
RLC    A
MOV    PB_8.A
  
```

To date, most banking terminals and other systems using the DES have needed special boards or peripheral controller chips just for the encryption decryption process, and still more hardware to form a serial bit stream for transmission (Figure 11.a). An 8051 solution could pack most of the entire system onto the one chip (Figure 11.b). The whole DES algorithm would require less than one-

fourth of the on-chip program memory, with the remaining bytes free for operating the banking terminal (or whatever) itself.

Moreover, since transmission and reception of data is performed through the on-board UART, the unencrypted data (plaintext) never even exists outside the microcomputer! Naturally, this would afford a high degree of security from data interception.

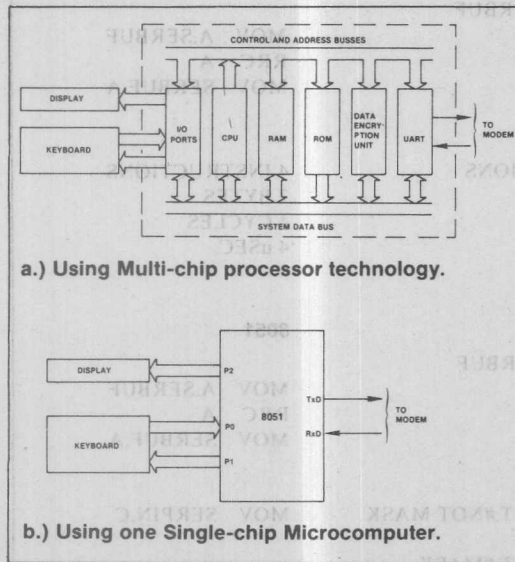


Figure 11. Secure Banking Terminal Block Diagram.

### Design Example #2 - Software Serial I/O

An exercise often imposed on beginning microcomputer students is to write a program simulating a UART. (See, for example, Application Notes AP24, AP29, and AP49.) Though doing this with the 8051 Family may appear to be a moot point (given that the hardware for a full UART is on-chip), it is still instructive to see how it would be done, and maintains a product line tradition.

As it turns out, the 8051 microcomputers can receive or transmit serial data via software very efficiently using the Boolean instruction set. Since any I/O pin may be a serial input or output, several serial links could be maintained at once.

Figures 12.a and 12.b show algorithms for receiving or transmitting a byte of data. (Another section of program would invoke this algorithm eight times, synchronizing it with a start bit, clock signal, software delay, or timer

interrupt.) Data is received by testing an input pin, setting the carry to the same state, shifting the carry into a data buffer, and saving the partial frame in internal RAM. Data is transmitted by shifting an output buffer through the carry, and generating each bit on an output pin.

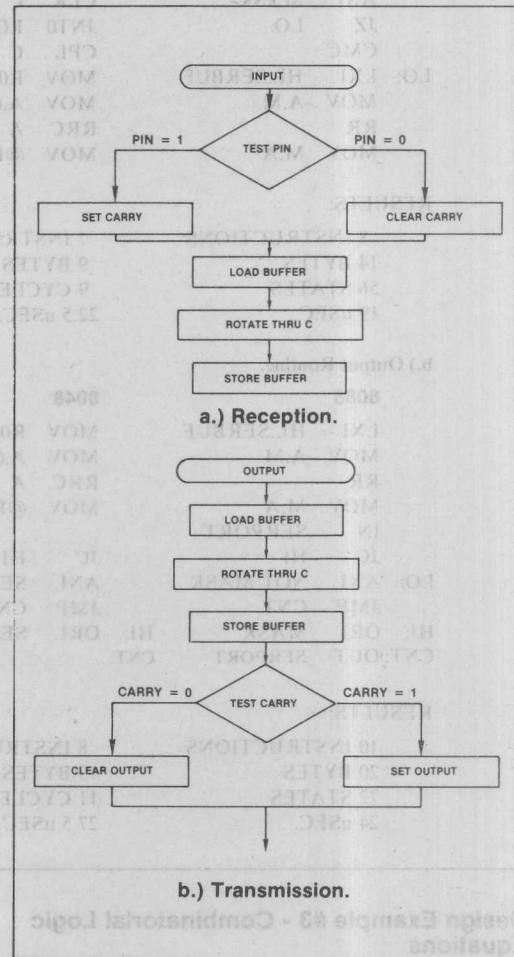


Figure 12. Serial I/O Algorithms.

A side-by-side comparison of the software for this common "bit-banging" application with three different microprocessor architectures is shown in Table 5.a and 5.b. The 8051 solution is more efficient than the others on every count!

**Table 5. Serial I/O Programs  
for Various Microprocessors.**

a.) Input Routine.	
<b>8085</b>	<b>8048</b>
IN SERPORT	CLR C
ANI MASK	JNT0 LO
JZ LO	CPL C
CMC	MOV R0,#SERBUF
LO: LXI HL,SERBUF	MOV A,@R0
MOV A,M	RRC A
RR	MOV @R0,A
MOV M,A	
RESULTS:	
8 INSTRUCTIONS	7 INSTRUCTIONS
14 BYTES	9 BYTES
56 STATES	9 CYCLES
19 uSEC.	22.5 uSEC.
b.) Output Routine.	
<b>8085</b>	<b>8048</b>
LXI HL,SERBUF	MOV R0,#SERBUF
MOV A,M	MOV A,@R0
RR	RRC A
MOV M,A	MOV @R0,A
IN SERPORT	JC HI
JC HI	ANL SERPRT,#NOT MASK
LO: ANI NOT MASK	JMP CNT
JMP CNT	HI: ORL SERPRT,#MASK
HI: ORI MASK	CNT:
CNT: OUT SERPORT	
RESULTS:	
10 INSTRUCTIONS	8 INSTRUCTIONS
20 BYTES	13 BYTES
72 STATES	11 CYCLES
24 uSEC.	27.5 uSEC.
<b>8051</b>	
MOV C,SERPIN	
MOV A,SERBUF	
RRC A	
MOV SERBUF,A	
RESULTS:	
4 INSTRUCTIONS	
7 BYTES	
4 CYCLES	
4 uSEC.	

### Design Example #3 - Combinatorial Logic Equations

Next we'll look at some simple uses for bit-test instructions and logical operations. (This example is also presented in Application Note AP-69.)

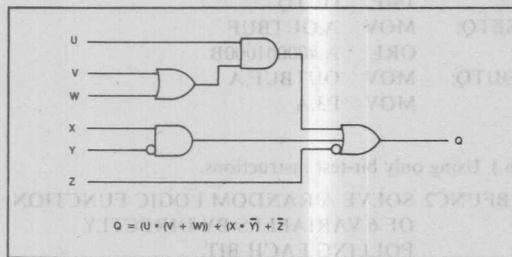
Virtually all hardware designers have solved complex functions using combinatorial logic. While the hardware involved may vary from relay logic, vacuum tubes, or TTL or to more esoteric technologies like fluidics, in each case the goal is the same: to solve a problem represented by a logical function of several Boolean variables.

Figure 13 shows TTL and relay logic diagrams for a function of the six variables U through Z. Each is a solution of the equation.

$$Q = (U \cdot (V + W)) + (X \cdot \bar{Y}) + \bar{Z}$$

Equations of this sort might be reduced using Karnaugh Maps or algebraic techniques, but that is not the purpose of this example. As the logic complexity increases, so does the difficulty of the reduction process. Even a minor change to the function equations as the design evolves would require tedious re-reduction from scratch.

Figure 13. Hardware Implementations of Boolean functions.



#### a.) Using TTL:

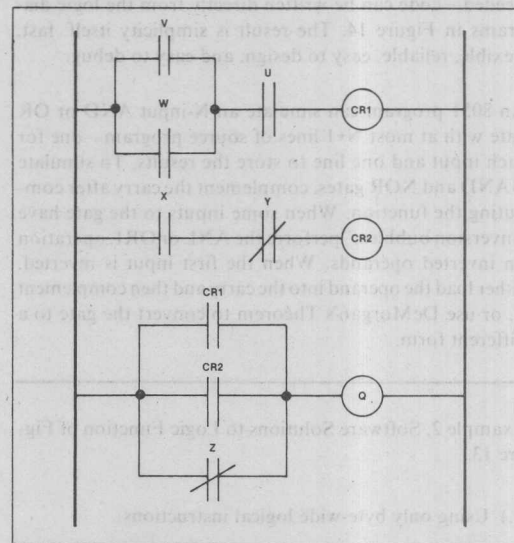
For the sake of comparison we will implement this function three ways, restricting the software to three proper subsets of the MCS-51™ instruction set. We will also assume that U and V are input pins from different input ports, W and X are status bits for two peripheral controllers, and Y and Z are software flags set up earlier in the program. The end result must be written to an output pin on some third port. The first two implementations follow the flow-chart shown in Figure 14. Program flow would embark on a route down a test-and-branch tree and leaves either the "True" or "Not True" exit ASAP — as soon as the proper result has been determined. These exits then rewrite the output port with the result bit respectively one or zero.

Other digital computers must solve equations of this type with standard word-wide logical instructions and conditional jumps. So for the first implementation, we won't use any generalized bit-addressing instructions. As we shall soon see, being constrained to such an instruction subset produces somewhat sloppy software solutions. MCS-51™ mnemonics are used in Example 2.a; other machines might further cloud the situation by requiring operation-specific mnemonics like INPUT, OUTPUT, LOAD, STORE, etc., instead of the MOV mnemonic used for all variable transfers in the 8051 instruction set.

The code which results is cumbersome and error prone. It would be difficult to prove whether the software worked for all input combinations in programs of this sort. Furthermore, execution time will vary widely with input data.

Thanks to the direct bit-test operations, a single instruction can replace each move/ mask/ conditional jump sequence in Example 2.a, but the algorithm would be equally convoluted (see Example 2.B). To lessen the confusion "a bit" each input variable is assigned a symbolic name.

A more elegant and efficient implementation (Example 2.c) strings together the Boolean ANL and ORL functions to generate the output function with straight-line code.



#### b.) Using Relay Logic:

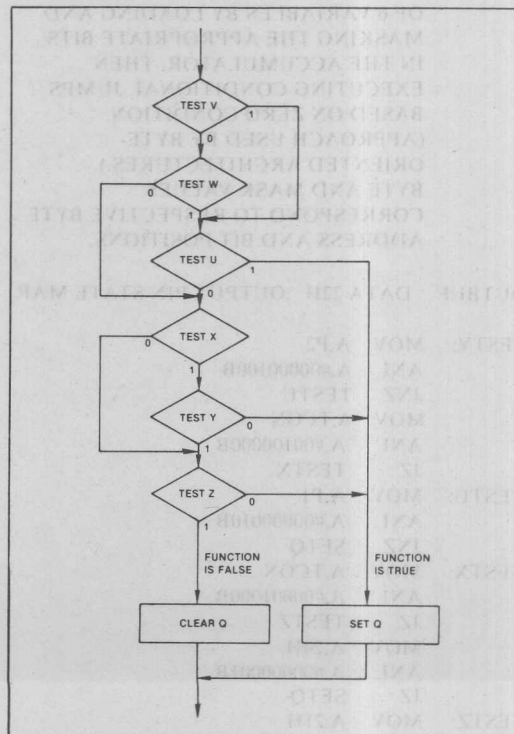


Figure 14. Flow chart for tree-branching algorithm.



When finished, the carry flag contains the result, which is simply copied out to the destination pin. No flow chart is needed—code can be written directly from the logic diagrams in Figure 14. The result is simplicity itself: fast, flexible, reliable, easy to design, and easy to debug.

An 8051 program can simulate an N-input AND or OR gate with at most N+1 lines of source program—one for each input and one line to store the results. To simulate NAND and NOR gates, complement the carry after computing the function. When some inputs to the gate have "inversion bubbles," perform the ANL or ORL operation on inverted operands. When the first input is inverted, either load the operand into the carry and then complement it, or use DeMorgan's Theorem to convert the gate to a different form.

Example 2. Software Solutions to Logic Function of Figure 13.

a.) Using only byte-wide logical instructions.

```
:BFUNC1 SOLVE RANDOM LOGIC FUNCTION
:      OF 6 VARIABLES BY LOADING AND
:      MASKING THE APPROPRIATE BITS
:      IN THE ACCUMULATOR, THEN
:      EXECUTING CONDITIONAL JUMPS
:      BASED ON ZERO CONDITION.
:      (APPROACH USED BY BYTE-
:      ORIENTED ARCHITECTURES.)
:      BYTE AND MASK VALUES
:      CORRESPOND TO RESPECTIVE BYTE
:      ADDRESS AND BIT POSITIONS.
```

OUTBUF DATA 22H :OUTPUT PIN STATE MAP

```
TESTV: MOV A,P2
      ANL A,#00000100B
      JNZ TESTU
      MOV A,TCON
      ANL A,#00100000B
      JZ TESTX
TESTU: MOV A,P1
      ANL A,#00000010B
      JNZ SETQ
TESTX: MOV A,TCON
      ANL A,#00001000B
      JZ TESTZ
      MOV A,20H
      ANL A,#00000001B
      JZ SETQ
TESTZ: MOV A,21H
      ANL A,#00000010B
      JZ SETQ
```

```
CLRQ: MOV A,OUTBUF
      ANL A,#11110111B
      JMP OUTQ
SETQ: MOV A,OUTBUF
      ORL A,#00001000B
OUTQ: MOV OUTBUF,A
      MOV P3,A
```

b.) Using only bit-test instructions.

```
:BFUNC2 SOLVE A RANDOM LOGIC FUNCTION
:      OF 6 VARIABLES BY DIRECTLY
:      POLLING EACH BIT.
:      (APPROACH USING MCS-51 UNIQUE
:      BIT-TEST INSTRUCTION CAPABILITY.)
:      SYMBOLS USED IN LOGIC DIAGRAM
:      ASSIGNED TO CORRESPONDING 8X51
:      BIT ADDRESSES.
```

```
      BIT P1.1
      BIT P2.2
      BIT TF0
      BIT IE1
      BIT 20H.0
      BIT 21H.1
      BIT P3.3
TEST_V: JB V.TEST_U
      JNB W.TEST_X
TEST_U: JB U.SET_Q
TEST_X: JNB X.TEST_Z
      JNB Y.SET_Q
TEST_Z: JNB Z.SET_Q
CLR_Q: CLR Q
      JMP NXTTST
SET_Q: SETB Q
NXTTST: : (CONTINUATION OF
:      PROGRAM)
```

c.) Using logical operations on Boolean variables.

```
:FUNC3 SOLVE A RANDOM LOGIC FUNCTION
:      OF 6 VARIABLES USING
:      STRAIGHT LINE LOGICAL
:      INSTRUCTIONS ON MCS-51 BOOLEAN
:      VARIABLES.
:
:      MOV C,V
:      ORL C,W :OUTPUT OF OR GATE
:      ANL C,U :OUTPUT OF TOP AND GATE
:      MOV F0,C :SAVE INTERMEDIATE STATE
:      MOV C,X
:      ANL C,Y :OUTPUT OF BOTTOM AND GATE
:      ORL C,F0 :INCLUDE VALUE SAVED ABOVE
:      ORL C,Z :INCLUDE LAST INPUT VARIABLE
:      MOV Q,C :OUTPUT COMPUTED RESULT
```

An upper-limit can be placed on the complexity of software to simulate a large number of gates by summing the total number of inputs and outputs. The *actual* total should be somewhat shorter, since calculations can be "chained," as shown above. The output of one gate is often the first input to another, bypassing the intermediate variable to eliminate two lines of source.

## Design Example #4 - Automotive Dashboard Functions

Now let's apply these techniques to designing the software for a complete controller system. This application is patterned after a familiar real-world application which isn't nearly as trivial as it might first appear: automobile turn signals.

Imagine the three position turn lever on the steering column as a single-pole, triple-throw toggle switch. In its central position all contacts are open. In the up or down positions contacts close causing corresponding lights in the rear of the car to blink. So far very simple.

Two more turn signals blink in the front of the car, and two others in the dashboard. All six bulbs flash when an emergency switch is closed. A thermo-mechanical relay (accessible under the dashboard in case it wears out) causes the blinking.

Applying the brake pedal turns the tail light filaments on constantly... unless a turn is in progress, in which case the blinking tail light is not affected. (Of course, the front turn signals and dashboard indicators are not affected by the brake pedal.) Table 6 summarizes these operating modes.

Table 6. Truth table for turn-signal operation.

INPUT SIGNALS				OUTPUT SIGNALS			
BRAKE SWITCH	EMERG. SWITCH	LEFT TURN SWITCH	RIGHT TURN SWITCH	LEFT FRONT & DASH	RIGHT FRONT & DASH	LEFT REAR	RIGHT REAR
0	0	0	0	OFF	OFF	OFF	OFF
0	0	0	1	OFF	BLINK	OFF	BLINK
0	0	1	0	BLINK	OFF	BLINK	OFF
0	1	0	0	BLINK	BLINK	BLINK	BLINK
0	1	0	1	BLINK	BLINK	BLINK	BLINK
0	1	1	0	BLINK	BLINK	BLINK	BLINK
1	0	0	0	OFF	OFF	ON	ON
1	0	0	1	OFF	BLINK	ON	BLINK
1	0	1	0	BLINK	OFF	BLINK	ON
1	1	0	0	BLINK	BLINK	ON	ON
1	1	0	1	BLINK	BLINK	ON	BLINK
1	1	1	0	BLINK	BLINK	BLINK	ON

But we're not done yet. Each of the exterior turn signal (but not the dashboard) bulbs has a second, somewhat dimmer filament for the parking lights. Figure 15 shows TTL circuitry which could control all six bulbs. The signals labeled "High Freq." and "Low Freq." represent two square-wave inputs. Basically, when one of the turn switches is closed or the emergency switch is activated the low frequency signal (about 1 Hz) is gated through to the appropriate dashboard indicator(s) and turn signal(s). The rear signals are also activated when the brake pedal is depressed provided a turn is not being made in the same direction. When the parking light switch is closed the higher frequency oscillator is gated to each front and rear turn signal, sustaining a low-intensity background level. (This is to eliminate the need for additional parking light filaments.)

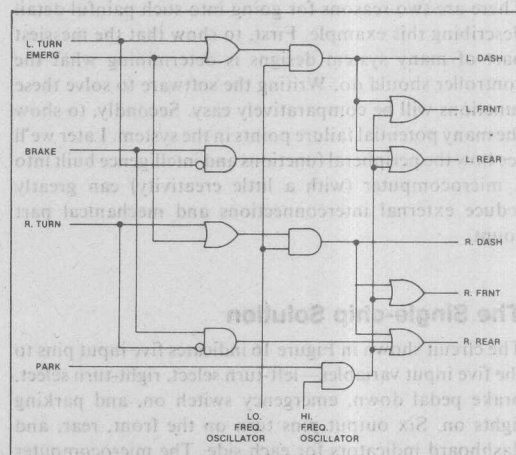


Figure 15. TTL logic implementation of automotive turn signals.

In most cars, the switching logic to generate these functions requires a number of multiple-throw contacts. As many as 18 conductors thread the steering column of some automobiles solely for turn-signal and emergency blinker functions. (The author discovered this recently to his astonishment and dismay when replacing the whole assembly because of one burned contact.)

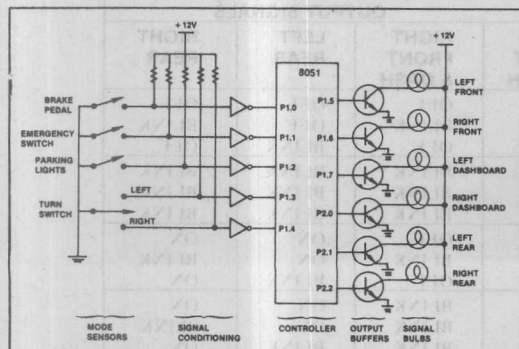
A multiple-conductor wiring harness runs to each corner of the car, behind the dash, up the steering column, and down to the blinker relay below. Connectors at each termination for each filament lead to extra cost and labor during construction, lower reliability and safety, and more costly repairs. And considering the system's present complexity, increasing its reliability or detecting failures would be quite difficult.

There are two reasons for going into such painful detail describing this example. First, to show that the messiest part of many system designs is determining what the controller should do. Writing the software to solve these functions will be comparatively easy. Secondly, to show the many potential failure points in the system. Later we'll see how the peripheral functions and intelligence built into a microcomputer (with a little creativity) can greatly reduce external interconnections and mechanical part count.

### The Single-chip Solution

The circuit shown in Figure 16 indicates five input pins to the five input variables—left-turn select, right-turn select, brake pedal down, emergency switch on, and parking lights on. Six output pins turn on the front, rear, and dashboard indicators for each side. The microcomputer implements all logical functions through software, which periodically updates the output signals as time elapses and input conditions change.

Figure 16. Microcomputer Turn-signal Connections.



Design Example #3 demonstrated that symbolic addressing with user-defined bit names makes code and documentation easier to write and maintain. Accordingly, we'll assign these I/O pins names for use throughout the program. (The format of this example will differ somewhat from the others. Segments of the overall program will be presented in sequence as each is described.)

```

:
: INPUT PIN DECLARATIONS:
: (ALL INPUTS ARE POSITIVE-TRUE LOGIC)
:
BRAKE BIT P1.0 : BRAKE PEDAL DEPRESSED
EMERG BIT P1.1 : EMERGENCY BLINKER
                  ACTIVATED
PARK BIT P1.2 : PARKING LIGHTS ON
L_TURN BIT P1.3 : TURN LEVER DOWN
R_TURN BIT P1.4 : TURN LEVER UP
:
: OUTPUT PIN DECLARATIONS:
:
L_FRNT BIT P1.5 : FRONT LEFT-TURN
                  INDICATOR
R_FRNT BIT P1.6 : FRONT RIGHT-TURN
                  INDICATOR
L_DASH BIT P1.7 : DASHBOARD LEFT-TURN
                  INDICATOR
R_DASH BIT P2.0 : DASHBOARD RIGHT-TURN
                  INDICATOR
L_REAR BIT P2.1 : REAR LEFT-TURN
                  INDICATOR
R_REAR BIT P2.2 : REAR RIGHT-TURN
                  INDICATOR
:

```

Another key advantage of symbolic addressing will appear further on in the design cycle. The locations of cable connectors, signal conditioning circuitry, voltage regulators, heat sinks, and the like all affect P.C. board layout. It's quite likely that the somewhat arbitrary pin assignment defined early in the software design cycle will prove to be less than optimum; rearranging the I/O pin assignment could well allow a more compact module, or eliminate costly jumpers on a single-sided board. (These considerations apply especially to automotive and other cost-sensitive applications needing single-chip controllers.) Since other architectures mask bytes or use "clever" algorithms to isolate bits by rotating them into the carry, re-routing an input signal (from bit 1 of port 1, for example, to bit 4 of port 3) could require extensive modifications throughout the software.

The Boolean Processor's direct bit addressing makes such changes absolutely trivial. The number of the port containing the pin is irrelevant, and masks and complex program structures are not needed. Only the initial Boolean varia-



```

; INTERRUPT RATE SUBDIVIDER
SUB_DIV DATA 20H
; HIGH-FREQUENCY OSCILLATOR BIT
HIFREQ BIT SUB_DIV.0
; LOW-FREQUENCY OSCILLATOR BIT
LO_FREQ BIT SUB_DIV.7
;
; ORG 0000H
JMP INIT
;
; ORG 100H
; PUT TIMER 0 IN MODE 1
INIT: MOV TMOD,#00000001B
; INITIALIZE TIMER REGISTERS
MOV TL0,#0
MOV TH0,#-16
; SUBDIVIDE INTERRUPT RATE BY 244
MOV SUB_DIV,#244
; ENABLE TIMER INTERRUPTS
SETB ET0
; GLOBALLY ENABLE ALL INTERRUPTS
SETB EA
; START TIMER
SETB TR0
; (CONTINUE WITH BACKGROUND PROGRAM)
; PUT TIMER 0 IN MODE 1
; INITIALIZE TIMER REGISTERS
; SUBDIVIDE INTERRUPT RATE BY 244
; ENABLE TIMER INTERRUPTS
; GLOBALLY ENABLE ALL INTERRUPTS
; START TIMER

```

ble declarations need to be changed; ASM51 automatically adjusts all addresses and symbolic references to the reassigned variables. The user is assured that no additional debugging or software verification will be required.

Timer 0 (one of the two on-chip timer/counters) replaces the thermo-mechanical blinker relay in the dashboard controller. During system initialization it is configured as a timer in mode 1 by setting the least significant bit of the timer mode register (TMOD). In this configuration the low-order byte (TL0) is incremented every machine cycle, overflowing and incrementing the high-order byte (TH0) every 256  $\mu$ Sec. Timer interrupt 0 is enabled so that a hardware interrupt will occur each time TH0 overflows. (For details of the numerous timer operating modes see the MCS-51™ User's Manual.)

An eight-bit variable in the bit-addressable RAM array will be needed to further subdivide the interrupts via software. The lowest-order bit of this counter toggles very

fast to modulate the parking lights; bit 7 will be "tuned" to approximately 1 Hz for the turn- and emergency-indicator blinking rate.

Loading TH0 with -16 will cause an interrupt after 4,096 msec. The interrupt service routine reloads the high-order byte of timer 0 for the next interval, saves the CPU registers likely to be affected on the stack, and then decrements SUB\_DIV. Loading SUB\_DIV with 244 initially and each time it decrements to zero will produce a 0.999 second period for the highest-order bit.

```

ORG 000BH ; TIMER 0 SERVICE VECTOR
MOV TH0,#-16
PUSH PSW
PUSH ACC
PUSH B
DJNZ SUB_DIV,T0SERV
MOV SUB_DIV,#244

```

The code to sample inputs, perform calculations, and update outputs—the real "meat" of the signal controller algorithm—may be performed either as part of the interrupt service routine or as part of a background program loop. The only concern is that it must be executed at least several dozen times per second to prevent parking light flickering. We will assume the former case, and insert the code into the timer 0 service routine.

First, notice from the logic diagram (Figure 15) that the subterm (PARK · H\_FREQ), asserted when the parking lights are to be on dimly, figures into four of the six output functions. Accordingly, we will first compute that term and save it in a temporary location named "DIM". The PSW contains two general purpose flags: F0, which corresponds to the 8048 flag of the same name, and PSW.1. Since The PSW has been saved and will be restored to its previous state after servicing the interrupt, we can use either bit for temporary storage.

```

DIM BIT PSW.1 ; DECLARE TEMP.
                STORAGE FLAG

```

```

;
; MOV C,PARK ; GATE PARKING
;              LIGHT SWITCH
; ANL HIFREQ ; WITH HIGH
;              FREQUENCY
;              SIGNAL
; MOV DIM,C ; AND SAVE IN
;              TEMP. VARIABLE.

```

This simple three-line section of code illustrates a remarkable point. The software indicates in very abstract terms exactly what function is being performed, independent of



the hardware configuration. The fact that these three bits include an input pin, a bit within a program variable, and a software flag in the PSW is totally invisible to the programmer.

Now generate and output the dashboard left turn signal.

```
MOV C,L_TURN      : SET CARRY IF
                   : TURN
ORL C,EMERG        : OR EMERGENCY
                   : SELECTED.
ANL C,I.O_FREQ     : GATE IN 1 HZ
                   : SIGNAL
MOV L,DASH.C       : AND OUTPUT TO
                   : DASHBOARD.
```

To generate the left front turn signal we only need to add the parking light function in F0. But notice that the function in the carry will also be needed for the rear signal. We can save effort later by saving its current state in F0.

```
MOV F0,C           : SAVE FUNCTION
                   : SO FAR.
ORL C,DIM           : ADD IN PARKING
                   : LIGHT FUNCTION
MOV L,FRNT.C       : AND OUTPUT TO
                   : TURN SIGNAL.
```

Finally, the rear left turn signal should also be on when the brake pedal is depressed, provided a left turn is not in progress.

```
MOV C,BRAKE        : GATE BRAKE
                   : PEDAL SWITCH
ANL C,L_TURN       : WITH TURN
                   : LEVER.
ORL C,F0           : INCLUDE TEMP.
                   : VARIABLE FROM
                   : DASH
ORL C,DIM           : AND PARKING
                   : LIGHT FUNCTION
MOV L,REAR.C       : AND OUTPUT TO
                   : TURN SIGNAL.
```

Now we have to go through a similar sequence for the right-hand equivalents to all the left-turn lights. This also gives us a chance to see how the code segments above look when combined.

```
MOV C,R_TURN      : SET CARRY IF
                   : TURN
ORL C,EMERG        : OR EMERGENCY
                   : SELECTED.
ANL C,I.O_FREQ     : IF SO, GATE IN 1
                   : HZ SIGNAL.
```

```
MOV R,DASH.C       : AND OUTPUT TO
                   : DASHBOARD.
MOV F0,C           : SAVE FUNCTION
                   : SO FAR.
ORL C,DIM           : ADD IN PARKING
                   : LIGHT FUNCTION
MOV R,FRNT.C       : AND OUTPUT TO
                   : TURN SIGNAL.
MOV C,BRAKE        : GATE BRAKE
                   : PEDAL SWITCH
ANL C,R_TURN       : WITH TURN
                   : LEVER.
ORL C,F0           : INCLUDE TEMP.
                   : VARIABLE FROM
                   : DASH
ORL C,DIM           : AND PARKING
                   : LIGHT FUNCTION
MOV R,REAR.C       : AND OUTPUT TO
                   : TURN SIGNAL.
```

(The perceptive reader may notice that simply rearranging the steps could eliminate one instruction from each sequence.)

Now that all six bulbs are in the proper states, we can return from the interrupt routine, and the program is finished. This code essentially needs to reverse the status saving steps at the beginning of the interrupt.

```
POP B              : RESTORE CPU
                   : REGISTERS.
POP ACC
POP PSW
RETI
```

**Program Refinements.** The luminescence of an incandescent light bulb filament is generally non-linear; the 50% duty cycle of HLFREQ may not produce the desired intensity. If the application requires, duty cycles of 25%, 75%, etc. are easily achieved by ANDing and ORing in additional low-order bits of SUB\_DIV. For example, 30 Hz signals of seven different duty cycles could be produced by considering bits 2-0 as shown in Table 7. The only software change required would be to the code which sets-up variable DIM:

```
MOV C,SUB_DIV.1    : START WITH 50
                   : PERCENT
ANL C,SUB_DIV.0    : MASK DOWN TO 25
                   : PERCENT
ORL C,SUB_DIV.2    : AND BUILD BACK TO
                   : 62 PERCENT
MOV DIM,C          : DUTY CYCLE FOR
                   : PARKING LIGHTS.
```

Table 7. Non-trivial Duty Cycles.

SUB_DIV BITS								DUTY CYCLES						
7	6	5	4	3	2	1	0	12.5%	25.0%	37.5%	50.0%	62.5%	75.0%	87.5%
X	X	X	X	X	0	0	0	OFF	OFF	OFF	OFF	OFF	OFF	OFF
X	X	X	X	X	0	0	1	OFF	OFF	OFF	OFF	OFF	OFF	ON
X	X	X	X	X	0	1	0	OFF	OFF	OFF	OFF	OFF	ON	ON
X	X	X	X	X	0	1	1	OFF	OFF	OFF	OFF	ON	ON	ON
X	X	X	X	X	1	0	0	OFF	OFF	OFF	ON	ON	ON	ON
X	X	X	X	X	1	0	1	OFF	OFF	ON	ON	ON	ON	ON
X	X	X	X	X	1	1	0	OFF	ON	ON	ON	ON	ON	ON
X	X	X	X	X	1	1	1	ON	ON	ON	ON	ON	ON	ON

Interconnections increase cost and decrease reliability. The simple buffered pin-per-function circuit in Figure 16 is insufficient when many outputs require higher-than-TTL drive levels. A lower-cost solution uses the 8051 serial port in the shift-register mode to augment I/O. In mode 0, writing a byte to the serial port data buffer (SBUF) causes the data to be output sequentially through the "RXD" pin while a burst of eight clock pulses is generated on the "TXD" pin. A shift register connected to these pins (Figure 17) will load the data byte as it is shifted out. A number of special peripheral driver circuits combining shift-register inputs with high drive level outputs have been introduced recently.

Cascading multiple shift registers end-to-end will expand the number of outputs even further. The data rate in the I/O expansion mode is one megabaud, or 8 usec. per byte. This is the mode which the serial port defaults to following a reset, so no initialization is required.

The software for this technique uses the B register as a "map" corresponding to the different output functions. The program manipulates these bits instead of the output pins. After all functions have been calculated the B register is shifted by the serial port to the shift-register driver. (While some outputs may glitch as data is shifted through them, at 1 Megabaud most people wouldn't notice. Some shift registers provide an "enable" bit to hold the output states while new data is being shifted in.)

This is where the earlier decision to address bits symbolically throughout the program is going to pay off. This major I/O restructuring is nearly as simple to implement as rearranging the input pins. Again, only the bit declarations need to be changed.

```

L_FRNT BIT B.0 : FRONT LEFT-TURN
INDICATOR
R_FRNT BIT B.1 : FRONT RIGHT-TURN
INDICATOR
L_DASH BIT B.2 : DASHBOARD LEFT-TURN
INDICATOR
R_DASH BIT B.3 : DASHBOARD RIGHT-TURN
INDICATOR

```

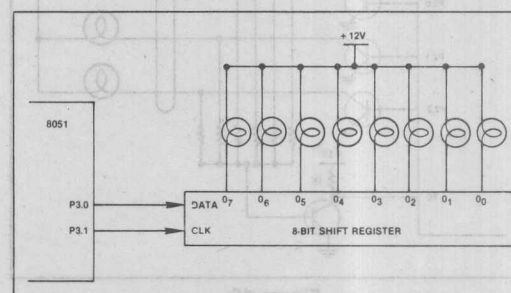


Figure 17. Output expansion using serial port.

```

L_REAR BIT B.4 : REAR LEFT-TURN
INDICATOR
R_REAR BIT B.5 : REAR RIGHT-TURN
INDICATOR

```

The original program to compute the functions need not change. After computing the output variables, the control map is transmitted to the buffered shift register through the serial port:

```
MOV SBUF,B : LOAD BUFFER AND TRANSMIT
```

The Boolean Processor solution holds a number of advantages over older methods. Fewer switches are required. Each is simpler, requiring fewer poles and lower current contacts. The flasher relay is eliminated entirely. Only six filaments are driven, rather than 10. The wiring harness is therefore simpler and less expensive—one conductor for each of the six lamps and each of the five sensor switches. The fewer conductors use far fewer connectors. The whole system is more reliable.

And since the system is much simpler it would be feasible to implement redundancy and/or fault detection on the four main turn indicators. Each could still be a standard double filament bulb, but with the filaments driven in parallel to tolerate single-element failures.

Even with redundancy, the lights will eventually fail. To handle this inescapable fact current or voltage sensing

circuits on each main drive wire can verify that each bulb and its high-current driver is functioning properly. Figure 18 shows one such circuit.

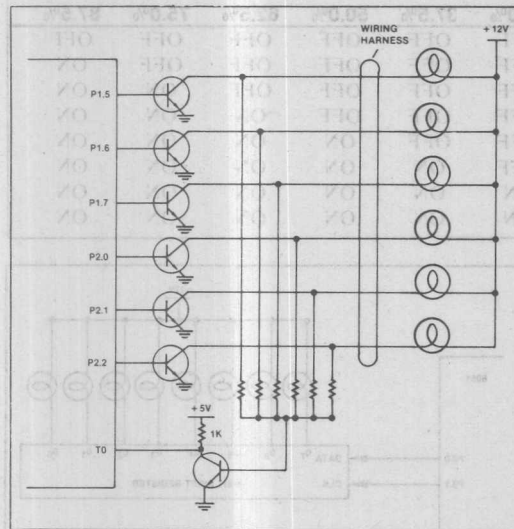


Figure 18.

Assume all of the lights are turned on except one; i.e., all but one of the collectors are grounded. For the bulb which is turned off, if there is continuity from +12 V through the bulb base and filament, the control wire, all connectors, and the P.C. board traces, and if the transistor is indeed not shorted to ground, then the collector will be pulled to +12 V. This turns on the base of Q8 through the corresponding resistor, and grounds the input pin, verifying that the bulb circuit is operational. The continuity of each circuit can be checked by software in this way.

Now turn *all* the bulbs on, grounding all the collectors. Q7 should be turned off, and the Test pin should be high. However, a control wire shorted to +12 V or an open-circuited drive transistor would leave one of the collectors at the higher voltage even now. This too would turn on Q7, indicating a different type of failure. Software could perform these checks once per second by executing the routine every time the software counter SUB\_DIV is reloaded by the interrupt routine.

```
DJNZ SUB_DIV,TOSERV
MOV SUB_DIV,#244      : RELOAD COUNTER
ORL P1,#11100000B     : SET CONTROL
                       : OUTPUTS HIGH
ORL P2,#00000111B
CLR I_FRNT            : FLOAT DRIVE
                       : COLLECTOR
JB T0,FAULT           : T0 SHOULD BE
                       : PULLED LOW
SETB I_FRNT           : PULL COLLECTOR
                       : BACK DOWN
```

```
CLR I_DASH
JB T0,FAULT
SETB I_DASH
CLR I_REAR
JB T0,FAULT
SETB I_REAR
CLR R_FRNT
JB T0,FAULT
SETB R_FRNT
CLR R_DASH
JB T0,FAULT
SETB R_DASH
CLR R_REAR
JB T0,FAULT
SETB R_REAR
```

```
: WITH ALL COLLECTORS GROUNDED, T0
: SHOULD BE HIGH
: IF SO, CONTINUE WITH INTERRUPT ROUTINE.
JB T0,TOSERV
FAULT: : ELECTRICAL FAILURE
: PROCESSING ROUTINE
: (LEFT TO READER'S
: IMAGINATION)
TOSERV: : CONTINUE WITH
: INTERRUPT PROCESSING
```

The complete assembled program listing is printed in Appendix A. The resulting code consists of 67 program statements, not counting declarations and comments, which assemble into 150 bytes of object code. Each pass through the service routine requires (coincidentally) 67 usec, plus 32 usec once per second for the electrical test. If executed every 4 msec as suggested this software would typically reduce the throughput of the background program by less than 2%.

Once a microcomputer has been designed into a system, new features suddenly become virtually free. Software could make the emergency blinkers flash alternately or at a rate faster than the turn signals. Turn signals could override the emergency blinkers. Adding more bulbs would allow multiple tail light sequencing and syncopation — true flash factor, so to speak.

## Design Example #5 - Complex Control Functions

Finally, we'll mix byte and bit operations to extend the use of 8051 into extremely complex applications.

Programmers can arbitrarily assign I/O pins to input and output functions only if the total does not exceed 32, which is insufficient for applications with a very large number of input variables. One way to expand the number of inputs is with a technique similar to multiplexed-keyboard scanning.

Figure 19 shows a block diagram for a moderately complex programmable industrial controller with the following characteristics:

- 64 input variable sensors;
- 12 output signals;
- Combinational and sequential logic computations;
- Remote operation with communications to a host processor via a high-speed full-duplex serial link;
- Two prioritized external interrupts;
- Internal real-time and time-of-day clocks.

While many microprocessors could be programmed to provide these capabilities with assorted peripheral support chips, an 8051 microcomputer needs **no** other integrated circuits!

The 64 input sensors are logically arranged as an 8x8 matrix. The pins of Port 1 sequentially enable each column of the sensor matrix; as each is enabled Port 0 reads in the state of each sensor in that column. An eight-byte block in bit-addressable RAM remembers the data as it is read in so that after each complete scan cycle there is an internal map of the current state of all sensors. Logic functions can then directly address the elements of the bit map.

The computer's serial port is configured as a nine-bit UART, transferring data at 17,000 bytes-per-second. The ninth bit may distinguish between address and data bytes.

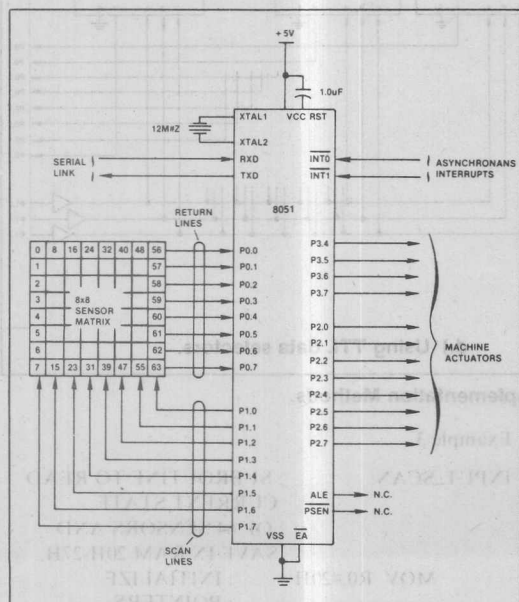


Figure 19. Block diagram of 64-input machine controller.

The 8051 serial port can be configured to detect bytes with the address bit set, automatically ignoring all others. Pins INT0 and INT1 are interrupts configured respectively as high-priority, falling-edge triggered and low-priority, low-level triggered. The remaining 12 I/O pins output TTL-level control signals to 12 actuators.

There are several ways to implement the sensor matrix circuitry, all logically similar. Figure 20.a shows one possibility. Each of the 64 sensors consists of a pair of simple switch contacts in series with a diode to permit multiple contact closures throughout the matrix.

The scan lines from Port 1 provide eight un-encoded active-high scan signals for enabling columns of the matrix. The return lines on rows where a contact is closed are pulled high and read as logic ones. Open return lines are pulled to ground by one of the 40 kohm resistors and are read as zeroes. (The resistor values must be chosen to ensure all return lines are pulled above the 2.0 V logic threshold, even in the worst-case, where all contacts in an enabled column are closed.) Since P0 is provided open-collector outputs and high-impedance MOS inputs its input loading may be considered negligible.

The circuits in Figures 20.b—20.d are variations on this theme. When input signals must be electrically isolated from the computer circuitry as in noisy industrial environments, phototransistors can replace the switch diode pairs **and** provide optical isolation as in Figure 20.b. Additional opto-isolators could also be used on the control output and special signal lines.

The other circuits assume that input signals are already at TTL levels. Figure 20.c uses octal three-state buffers enabled by active-low scan signals to gate eight signals onto Port 0. Port 0 is available for memory expansion or peripheral chip interfacing between sensor matrix scans. Eight-to-one multiplexers in Figure 20.d select one of eight inputs for each return line as determined by encoded address bits output on three pins of Port 1. (Five more output pins are thus freed for more control functions.) Each output can drive at least one standard TTL or up to 10 low-power TTL loads without additional buffering.

Going back to the original matrix circuit, Figure 21 shows the method used to scan the sensor matrix. Two complete bit maps are maintained in the bit-addressable region of the RAM: one for the current state and one for the previous state read for each sensor. If the need arises, the program could then sense input transitions and/or debounce contact closures by comparing each bit with its earlier value.



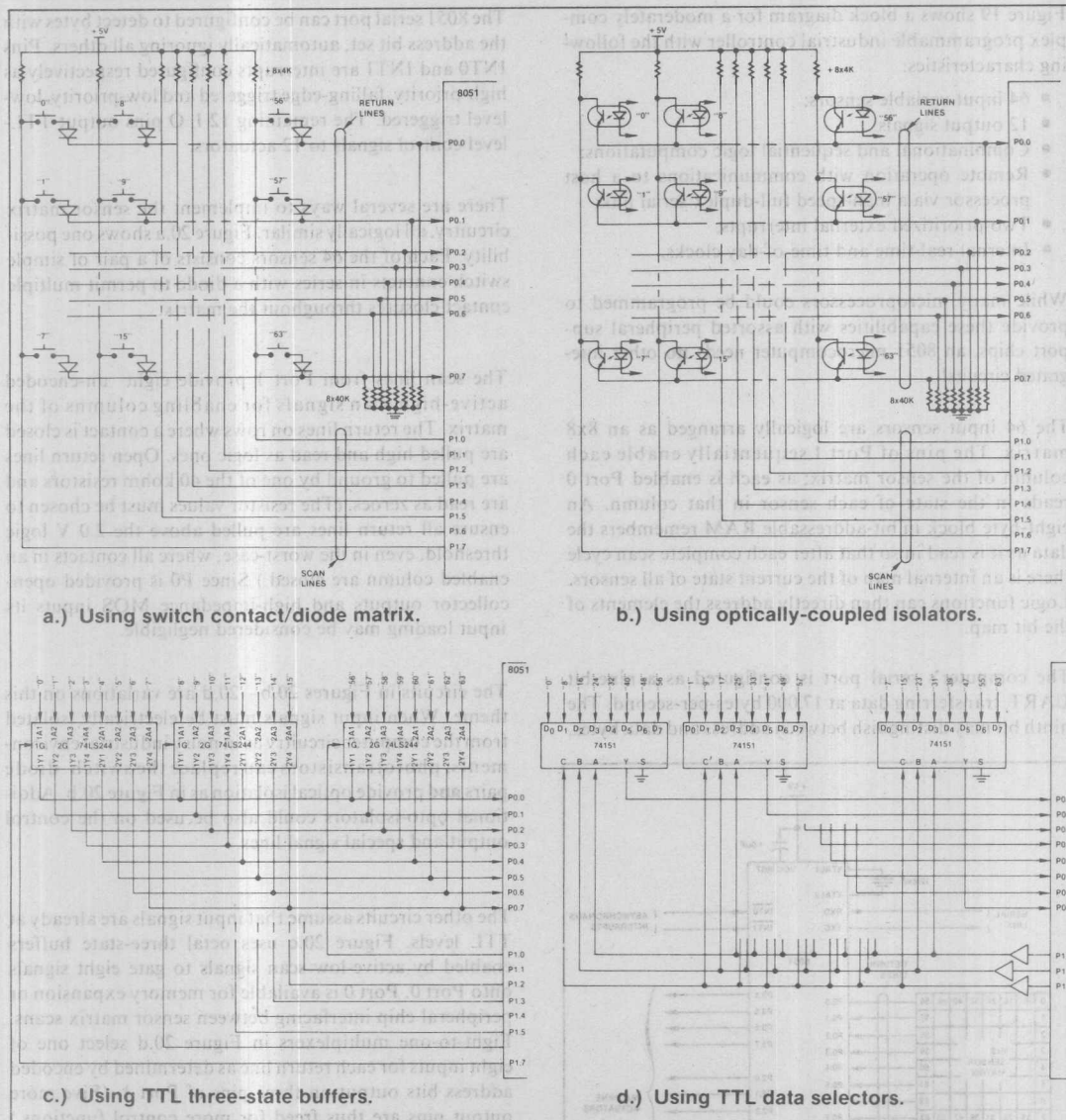


Figure 20. Sensor Matrix Implementation Methods.

Example 3.

```

INPUT_SCAN:      : SUBROUTINE TO READ
                  : CURRENT STATE
                  : OF 64 SENSORS AND
                  : SAVE IN RAM 20H-27H.
MOV R0,#20H      : INITIALIZE
                  : POINTERS
MOV R1,#28H      : FOR BIT MAP
                  : BASES.

```

The code in Example 3 implements the scanning algorithm for the circuits in Figure 20.a. Each column is enabled by setting a single bit in a field of zeroes. The bit maps are positive logic; ones represent contacts that are closed or isolators turned on.

```

MOV A,#80H      : SET FIRST BIT IN
                  ACC.
SCAN: MOV P1.A   : OUTPUT TO SCAN
                  LINES.
RR A            : SHIFT TO ENABLE
                  NEXT COLUMN
                  NEXT.
MOV R2.A        : REMEMBER CUR-
                  RENT SCAN
                  POSITION.
MOV A,P0        : READ RETURN
                  LINES.
XCH A,@R0       : SWITCH WITH
                  PREVIOUS MAP
                  BITS.
MOV @R1.A       : SAVE PREVIOUS
                  STATE AS WELL.
INC R0          : BUMP POINTERS.
INC R1
MOV A,R2        : RELOAD SCAN LINE
                  MASK
JNB ACC.7,SCAN  : LOOP UNTIL ALL
                  EIGHT COLUMNS
                  READ.
RET

```

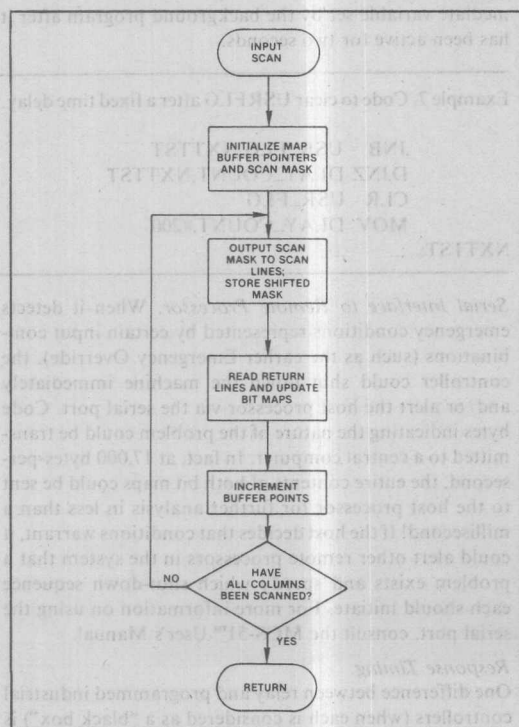


Figure 21. Flowchart for reading in sensor matrix.

What happens after the sensors have been scanned depends on the individual application. Rather than inventing some artificial design problem, software corresponding to commonplace logic elements will be discussed.

**Combinatorial Output Variables.** An output variable which is a simple (or not so simple) combinational function of several input variables is computed in the spirit of Design Example 3. All 64 inputs are represented in the bit maps; in fact, the sensor numbers in Figure 20 correspond to the absolute bit addresses in RAM! The code in Example 4 activates an actuator connected to P2.2 when sensors 12, 23, and 34 are closed and sensors 45 and 56 are open.

Example 4.

Simple Combinatorial Output Variables.

```

: SET P2.2 = (12) (23) (34) ( 45) ( 56)
MOV C,12
ANI C,23
ANI C,34
ANI C, 45
ANI C, 56
MOV P2.2,C

```

**Intermediate Variables.** The examination of a typical relay-logic ladder diagram will show that many of the rungs control *not* outputs but rather relays whose contacts figure into the computation of other functions. In effect, these relays indicate the state of intermediate variables of a computation.

The MCS-51™ solution can use any directly addressable bit for the storage of such intermediate variables. Even when all 128 bits of the RAM array are dedicated (to input bit maps in this example), the accumulator, PSW, and B register provide 18 additional flags for intermediate variables.

For example, suppose switches 0 through 3 control a safety interlock system. Closing any of them should deactivate certain outputs. Figure 22 is a ladder diagram for this situation. The interlock function could be recomputed for every output affected, or it may be computed once and saved (as implied by the diagram). As the program proceeds this bit can qualify each output.

Example 5. Incorporating Override signal into actuator outputs.

```

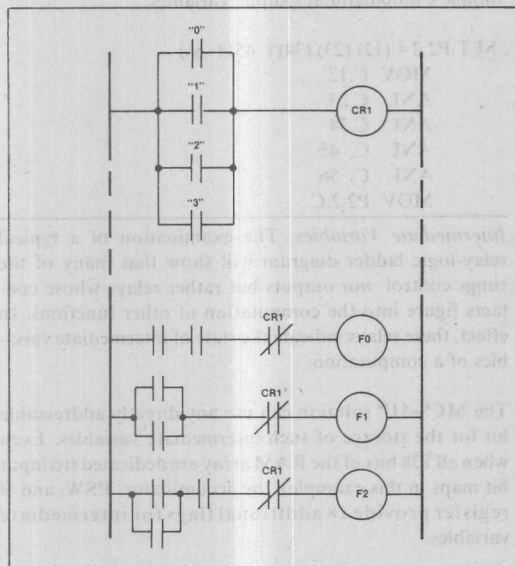
CALL INPUT_SCAN
MOV C,0
ORL C,1
ORL C,2
ORL C,3
MOV F0,C
:
:
:

```

```

: COMPUTE FUNCTION 0
: ANI C, F0
: MOV PI.0.C
: .....
: COMPUTE FUNCTION 1
: ANI C, F0
: MOV PI.1.C
: .....
: COMPUTE FUNCTION 2
: ANI C, F0
: MOV PI.2.C
: .....

```



**Figure 22. Ladder diagram for output override circuitry.**

**Latching Relays.** A latching relay can be forced into either the ON or OFF state by two corresponding input signals, where it will remain until forced onto the opposite state—analogue to a TTL Set / Reset flip-flop. The relay is used as an intermediate variable for other calculations. In the previous example, the emergency condition could be remembered and remain active until an “emergency cleared” button is pressed.

Any flag or addressable bit may represent a latching relay with a few lines of code (see Example 6).

Example 6. Simulating a latching relay.

```

: I_SET SET FLAG 0 IF C=1
: I_SET ORL C,F0
: MOV F0.C
: .....
: I_RSET RESET FLAG 0 IF C=1
: I_RSET CPS C
: ANI C,F0
: MOV F0.C
: .....

```

**Time Delay Relays.** A time delay relay does not respond to an input signal until it has been present (or absent) for some predefined time. For example, a ballast or load resistor may be switched in series with a D.C. motor when it is first turned on, and shunted from the circuit after one second. This sort of time delay may be simulated by an interrupt routine driven by one of the two 8051 timer/counters. The procedure followed by the routine depends heavily on the details of the exact function needed; time-outs or time delays with resettable or non-resettable inputs are possible. If the interrupt routine is executed every 10 milliseconds the code in Example 7 will clear an intermediate variable set by the background program after it has been active for two seconds.

Example 7. Code to clear USRFLG after a fixed time delay.

```

: JNB USR_FLG,NXTTST
: DJNZ DLAY_COUNT,NXTTST
: CLR USR_FLG
: MOV DLAY_COUNT,#200
: NXTTST: ...

```

**Serial Interface to Remote Processor.** When it detects emergency conditions represented by certain input combinations (such as the earlier Emergency Override), the controller could shut down the machine immediately and/or alert the host processor via the serial port. Code bytes indicating the nature of the problem could be transmitted to a central computer. In fact, at 17,000 bytes-per-second, the entire contents of both bit maps could be sent to the host processor for further analysis in less than a millisecond! If the host decides that conditions warrant, it could alert other remote processors in the system that a problem exists and specify which shut-down sequence each should initiate. For more information on using the serial port, consult the MCS-51™ User's Manual.

#### Response Timing.

One difference between relay and programmed industrial controllers (when each is considered as a “black box”) is their respective reaction times to input changes. As reflected by a ladder diagram, relay systems contain a

large number of "rungs" operating in parallel. A change in input conditions will begin propagating through the system immediately, possibly affecting the output state within milliseconds.

Software, on the other hand, operates sequentially. A change in input states will not be detected until the next time an input scan is performed, and will not affect the outputs until that section of the program is reached. For that reason the raw speed of computing the logical functions is of extreme importance.

Here the Boolean processor pays off. *Every instruction mentioned in this Note* completes in one or two microseconds—the *minimum* instruction execution time for many other microcontrollers! A ladder diagram containing a hundred rungs, with an average of four contacts per rung can be replaced by approximately five hundred lines of software. A complete pass through the entire matrix scanning routine and all computations would require about a millisecond; less than the time it takes for most relays to change state.

A programmed controller which simulates each Boolean function with a subroutine would be less efficient by at least an order of magnitude. Extra software is needed for the simulation routines, and each step takes longer to execute for three reasons: several byte-wide logical instructions are executed per user program step (rather than one Boolean operation); most of those instructions take longer to execute with microprocessors performing multiple off-chip accesses; and calling and returning from the various subroutines requires overhead for stack operations.

In fact, the speed of the Boolean Processor solution is likely to be much faster than the system requires. The CPU might use the time left over to compute feedback parameters, collect and analyze execution statistics, perform system diagnostics, and so forth.

### Additional functions and uses.

With the building-block basics mentioned above many more operations may be synthesized by short instruction sequences.

**Exclusive-OR.** There are no common mechanical devices or relays analogous to the Exclusive-OR operation, so this instruction was omitted from the Boolean Processor. However, the Exclusive-OR or Exclusive-NOR operation may be performed in two instructions by conditionally complementing the carry or a Boolean variable based on the state of any other testable bit.

**EXCLUSIVE-OR FUNCTION IMPOSED ON CARRY USING F0 IS INPUT VARIABLE.**

```
XOR_F0: JNB  F0,XORCNT ; ("JB" FOR X-NOR)
        CPL  C
```

XORCNT: ... ..

**XCH.** The contents of the carry and some other bit may be exchanged (switched) by using the accumulator as temporary storage. Bits can be moved into and out of the accumulator simultaneously using the Rotate-through-carry instructions, though this would alter the accumulator data.

**EXCHANGE CARRY WITH USRFLG**

```
XCHBIT: RLC  A
        MOV  C,USR_FLG
        RRC  A
        MOV  USR_FLG,C
        RLC  A
```

**Extended Bit Addressing.** The 8051 can directly address 144 general-purpose bits for all instructions in Figure 3.b. Similar operations may be extended to any bit anywhere on the chip with some loss of efficiency.

The logical operations AND, OR, and Exclusive-OR are performed on byte variables using six different addressing modes, one of which lets the source be an immediate mask, and the destination any directly addressable byte. Any bit may thus be set, cleared, or complemented with a three-byte, two-cycle instruction if the mask has all bits but one set or cleared.

Byte variables, registers, and indirectly addressed RAM may be moved to a bit addressable register (usually the accumulator) in one instruction. Once transferred, the bits may be tested with a conditional jump, allowing any bit to be polled in 3 microseconds—still much faster than most architectures—or used for logical calculations. (This technique can also simulate additional bit addressing modes with byte operations.)

**Parity of bytes or bits.** The parity of the current accumulator contents is always available in the PSW, from whence it may be moved to the carry and further processed. Error-correcting Hamming codes and similar applications require computing parity on groups of isolated bits. This can be done by conditionally complementing the carry flag based on those bits or by gathering the bits into the accumulator (as shown in the DES example) and then testing the parallel parity flag.

**Multiple byte shift and CRC codes.**

Though the 8051 serial port can accommodate eight- or nine-bit data transmissions, some protocols involve much



longer bit streams. The algorithms presented in Design Example 2 can be extended quite readily to 16 or more bits by using multi-byte input and output buffers.

Many mass data storage peripherals and serial communications protocols include Cyclic Redundancy (CRC) codes to verify data integrity. The function is generally computed serially by hardware using shift registers and Exclusive-OR gates, but it can be done with software. As each bit is received into the carry, appropriate bits in the multi-byte data buffer are conditionally complemented based on the incoming data bit. When finished, the CRC register contents may be checked for zero by ORing the two bytes in the accumulator.

#### 4. SUMMARY

A truly unique facet of the Intel MCS-51™ microcomputer family design is the collection of features optimized for the one-bit operations so often desired in real-world, real-time control applications. Included are 17 special instructions, a Boolean accumulator, implicit and direct addressing modes, program and mass data storage, and many I/O options. These are the world's first single-chip microcomputers able to efficiently manipulate, operate on, and transfer either bytes or individual bits as data.

This Application Note has detailed the information needed by a microcomputer system designer to make full use of these capabilities. Five design examples were used to contrast the solutions allowed by the 8051 and those required by previous architectures. Depending on the individual application, the 8051 solution will be easier to design, more reliable to implement, debug, and verify, use less program memory, and run up to an order of magnitude faster than the same function implemented on previous digital computer architectures.

Combining byte- and bit-handling capabilities in a single microcomputer has a strong synergistic effect: the power of the result exceeds the power of byte- and bit-processors laboring individually. Virtually all user applications will benefit in some ways from this duality. Data intensive applications will use bit addressing for test pin monitoring or program control flags; control applications will use byte manipulation for parallel I/O expansion or arithmetic calculations.

It is hoped that these design examples give the reader an appreciation of these unique features and suggest ways to exploit them in his or her own application.

The logical operations AND, OR, and Exclusive-OR are performed on bit strings against different addressing modes, one of which sets the source to be complemented, and the destination may directly address the data. Any bit may thus be set, cleared, or complemented with a three-byte, two-cycle instruction if the mask has all bits but one set or cleared.

Byte variables, registers, and indirectly addressed RAM may be moved to a bit addressable register (usually the accumulator) in one instruction. Once transferred, the bit may be tested with a conditional jump, allowing execution to be polled in 3 microseconds—still much faster than most architectures—or used for logical calculations. This technique can also simulate additional bit addressing modes with byte operations.

Part of a byte or bit. The carry of the current instruction contents is always available in the PSW, from which it may be moved to the carry and further processed. Error-correcting checksum codes and similar applications require comparing parity on groups of bits. This can be done by conditionally complementing the carry flag based on those bits or by gathering the bits into the accumulator (shown in the DES example) and then testing the parity flag.

Multiple bit shift and CRC codes. Though the 8051 serial port can accommodate eight or nine-bit data transmissions, some protocols involve multi-

In fact, the speed of the Boolean Processor solution is likely to be much faster than the system requires. The CPU might use the time left over to compute feedback parameters, collect and analyze execution statistics, perform system diagnosis, and so forth.

#### Additional functions and uses.

With the building-block basics mentioned above, many more operations may be synthesized by short instruction sequences.

Exclusive-OR. There are no common mechanical devices or relay analogs to the Exclusive-OR operation, so this instruction was omitted from the Boolean Processor. However, the Exclusive-OR or Exclusive-NOR operation may be performed in two instructions by conditionally complementing the carry of a Boolean variable based on the state of any other testable bit.

ISIS-II MCS-51 MACRO ASSEMBLER V1.0  
 OBJECT MODULE PLACED IN :FO:AP70.HEX  
 ASSEMBLER INVOKED BY : f1:asm51 ap70 src date(328)  
 LOC OBJ LINE SOURCE

```

0000 0000 1 $XREF TITLE(AP-70 APPENDIX)
0000 0000 2 ; *****
0000 0000 3 ;
0000 0000 4 ; THE FOLLOWING PROGRAM USES THE BOOLEAN INSTRUCTION SET
0000 0000 5 ; OF THE INTEL 8051 MICROCOMPUTER TO PERFORM A NUMBER OF
0000 0000 6 ; AUTOMOTIVE DASHBOARD CONTROL FUNCTIONS RELATING TO
0000 0000 7 ; TURN SIGNAL CONTROL, EMERGENCY BLINKERS, BRAKE LIGHT
0000 0000 8 ; CONTROL, AND PARKING LIGHT OPERATION.
0000 0000 9 ; THE ALGORITHMS AND HARDWARE ARE DESCRIBED IN DESIGN
0000 0000 10 ; EXAMPLE #4 OF INTEL APPLICATION NOTE AP-70,
0000 0000 11 ; "USING THE INTEL MCS-51(TM)
0000 0000 12 ; BOOLEAN PROCESSING CAPABILITIES"
0000 0000 13 ; *****
0000 0000 14 ;
0000 0000 15 ; INPUT PIN DECLARATIONS:
0000 0000 16 ; (ALL INPUTS ARE POSITIVE-TRUE LOGIC)
0000 0000 17 ; INPUTS ARE HIGH WHEN RESPECTIVE SWITCH CONTACT IS CLOSED.)
0000 0000 18 ;
0000 0000 19 ;
0000 0090 20 BRAKE BIT P1.0 ; BRAKE PEDAL DEPRESSED
0000 0091 21 EMERG BIT P1.1 ; EMERGENCY BLINKER ACTIVATED
0000 0092 22 PARK BIT P1.2 ; PARKING LIGHTS ON
0000 0093 23 L_TURN BIT P1.3 ; TURN LEVER DOWN
0000 0094 24 R_TURN BIT P1.4 ; TURN LEVER UP
0000 0095 25 ;
0000 0096 26 ; OUTPUT PIN DECLARATIONS:
0000 0097 27 ; (ALL OUTPUTS ARE POSITIVE TRUE LOGIC.
0000 0098 28 ; BULB IS TURNED ON WHEN OUTPUT PIN IS HIGH.)
0000 0099 29 ;
0000 0095 30 L_FRNT BIT P1.5 ; FRONT LEFT-TURN INDICATOR
0000 0096 31 R_FRNT BIT P1.6 ; FRONT RIGHT-TURN INDICATOR
0000 0097 32 L_DASH BIT P1.7 ; DASHBOARD LEFT-TURN INDICATOR
0000 00A0 33 R_DASH BIT P2.0 ; DASHBOARD RIGHT-TURN INDICATOR
0000 00A1 34 L_REAR BIT P2.1 ; REAR LEFT-TURN INDICATOR
0000 00A2 35 R_REAR BIT P2.2 ; REAR RIGHT-TURN INDICATOR
0000 00A3 36 ;
0000 00A3 37 S_FAIL BIT P2.3 ; ELECTRICAL SYSTEM FAULT INDICATOR
0000 00A3 38 ;
0000 00A3 39 ; INTERNAL VARIABLE DEFINITIONS:
0000 00A3 40 ;
0000 0020 41 SUB_DIV DATA 20H ; INTERRUPT RATE SUBDIVIDER
0000 0000 42 HI_FREQ BIT SUB_DIV.0 ; HIGH-FREQUENCY OSCILLATOR BIT
0000 0007 43 LO_FREQ BIT SUB_DIV.7 ; LOW-FREQUENCY OSCILLATOR BIT
0000 0000 44 ;
0000 00D1 45 DIM BIT PSW.1 ; PARKING LIGHTS ON FLAG
0000 00D1 46 ;
0000 00D1 47 ; *****
0000 00D1 48 +1 $EJECT

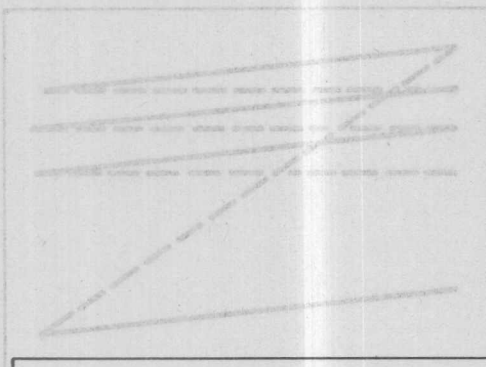
```

LOC	OBJ	LINE	SOURCE
		49	ORG 0000H ; RESET VECTOR THIS ON L740
0000	020040	50	LJMP INIT
		51	
000B	758CF0	52	ORG 000BH ; TIMER 0 SERVICE VECTOR
000E	C0D0	53	MOV TH0, #-16 ; HIGH TIMER BYTE ADJUSTED TO CONTROL INT. RATE
0010	0154	54	PUSH PSW ; EXECUTE CODE TO SAVE ANY REGISTERS USED BELOW
		55	AJMP UPDATE ; (CONTINUE WITH REST OF ROUTINE)
		56	
0040		57	ORG 0040H
0040	758A00	58	INIT: MOV TLO, #0 ; ZERO LOADED INTO LOW-ORDER BYTE AND
0043	758CF0	59	MOV TH0, #-16 ; -16 IN HIGH-ORDER BYTE GIVES 4 MSEC PERIOD
0046	758961	60	MOV TMOD, #01100001B ; 8-BIT AUTO RELOAD COUNTER MODE FOR TIMER 1,
		61	16-BIT TIMER MODE FOR TIMER 0 SELECTED
0049	7520F4	62	MOV SUB_DIV, #244 ; SUBDIVIDE INTERRUPT RATE BY 244 FOR 1 HZ
004C	D2A9	63	SETB ET0 ; USE TIMER 0 OVERFLOWS TO INTERRUPT PROGRAM
004E	D2AF	64	SETB EA ; CONFIGURE IE TO GLOBALLY ENABLE INTERRUPTS
0050	D28C	65	SETB TRO ; KEEP INSTRUCTION CYCLE COUNT UNTIL OVERFLOW
0052	80FE	66	SJMP \$ ; LOOPS ON WHEN START BACKGROUND PROGRAM EXECUTION
		67	
		68	
0054	D52038	69	UPDATE: DJNZ SUB_DIV, TOSERV ; EXECUTE SYSTEM TEST ONLY ONCE PER SECOND
0057	7520F4	70	MOV SUB_DIV, #244 ; GET VALUE FOR NEXT ONE SECOND DELAY AND
		71	GO THROUGH ELECTRICAL SYSTEM TEST CODE:
005A	4390E0	72	ORL P1, #11100000B ; SET CONTROL OUTPUTS HIGH
005D	43A007	73	ORL P2, #00000111B ; SET CONTROL OUTPUTS HIGH
0060	C295	74	CLR L_FRNT ; BYTE FLOAT DRIVE COLLECTOR
0062	20B428	75	JB TO, FAULT ; TO SHOULD BE PULLED LOW
0065	D295	76	SETB L_FRNT ; PULL COLLECTOR BACK DOWN
0067	C297	77	CLR L_DASH ; REPEAT SEQUENCE FOR L_DASH,
0069	20B421	78	JB TO, FAULT
006C	D297	79	SETB L_DASH
006E	C2A1	80	CLR L_REAR ;
0070	20B41A	81	JB TO, FAULT
0073	D2A1	82	SETB L_REAR ;
0075	C296	83	CLR R_FRNT ;
0077	20B413	84	JB TO, FAULT ;
007A	D296	85	SETB R_FRNT ;
007C	C2A0	86	CLR R_DASH ;
007E	20B40C	87	JB TO, FAULT ;
0081	D2A0	88	SETB R_DASH ;
0083	C2A2	89	CLR L_REAR ;
0085	20B405	90	JB TO, FAULT ;
0088	D2A2	91	SETB L_REAR ;
		92	
		93	
		94	WITH ALL COLLECTORS GROUNDED, TO SHOULD BE HIGH
		95	IF SO, CONTINUE WITH INTERRUPT ROUTINE.
		96	
008A	20B402	97	JB TO, TOSERV
008D	B2A3	98	FAULT: CPL S_FAIL ; ELECTRICAL FAILURE PROCESSING ROUTINE
		99	TOGGLE INDICATOR ONCE PER SECOND)
		99	+\$EJECT

LOC	OBJ	LINE	SOURCE
		100	CONTINUE WITH INTERRUPT PROCESSING.
		101	
		102	1) COMPUTE LOW BULB INTENSITY WHEN PARKING LIGHTS ARE ON.
		103	
008F	A201	104	TOSERV: MOV C.SUB_DIV.1 ; START WITH 50 PERCENT,
0091	8200	105	ANL C.SUB_DIV.0 ; MASK DOWN TO 25 PERCENT.
0093	7202	106	ORL C.SUB_DIV.2 ; BUILD BACK TO 62.5 PERCENT.
0095	8292	107	ANL C.PARK ; GATE WITH PARKING LIGHT SWITCH.
0097	92D1	108	MOV DIM,C ; AND SAVE IN TEMP. VARIABLE.
		109	
		110	2) COMPUTE AND OUTPUT LEFT-HAND DASHBOARD INDICATOR.
		111	
0099	A293	112	MOV C.L_TURN ; SET CARRY IF TURN
009B	7291	113	ORL C.EMERG ; OR EMERGENCY SELECTED.
009D	8207	114	ANL C.LO_FREQ ; IF SO, GATE IN 1 HZ SIGNAL
009F	9297	115	MOV L_DASH,C ; AND OUTPUT TO DASHBOARD.
		116	
		117	3) COMPUTE AND OUTPUT LEFT-HAND FRONT TURN SIGNAL.
		118	
00A1	92D5	119	MOV FO,C ; SAVE FUNCTION SO FAR.
00A3	72D1	120	ORL C.DIM ; ADD IN PARKING LIGHT FUNCTION
00A5	9295	121	MOV L_FRNT,C ; AND OUTPUT TO TURN SIGNAL.
		122	
		123	4) COMPUTE AND OUTPUT LEFT-HAND REAR TURN SIGNAL.
		124	
00A7	A290	125	MOV C.BRAKE ; GATE BRAKE PEDAL SWITCH
00A9	B093	126	ANL C./L_TURN ; WITH TURN LEVER.
00AB	72D5	127	ORL C.FO ; INCLUDE TEMP. VARIABLE FROM DASH
00AD	72D1	128	ORL C.DIM ; AND PARKING LIGHT FUNCTION
00AF	92A1	129	MOV L_REAR,C ; AND OUTPUT TO TURN SIGNAL.
		130	
		131	5) REPEAT ALL OF ABOVE FOR RIGHT-HAND COUNTERPARTS.
		132	
00B1	A294	133	MOV C.R_TURN ; SET CARRY IF TURN
00B3	7291	134	ORL C.EMERG ; OR EMERGENCY SELECTED.
00B5	8207	135	ANL C.LO_FREQ ; IF SO, GATE IN 1 HZ SIGNAL
00B7	92A0	136	MOV R_DASH,C ; AND OUTPUT TO DASHBOARD.
00B9	92D5	137	MOV FO,C ; SAVE FUNCTION SO FAR.
00BB	72D1	138	ORL C.DIM ; ADD IN PARKING LIGHT FUNCTION
00BD	9296	139	MOV R_FRNT,C ; AND OUTPUT TO TURN SIGNAL.
00BF	A290	140	MOV C.BRAKE ; GATE BRAKE PEDAL SWITCH
00C1	B094	141	ANL C./R_TURN ; WITH TURN LEVER.
00C3	72D5	142	ORL C.FO ; INCLUDE TEMP. VARIABLE FROM DASH
00C5	72D1	143	ORL C.DIM ; AND PARKING LIGHT FUNCTION
00C7	92A2	144	MOV R_REAR,C ; AND OUTPUT TO TURN SIGNAL.
		145	
		146	RESTORE STATUS REGISTER AND RETURN.
		147	
00C9	D0D0	148	POP PSW ; RESTORE PSW
00CB	32	149	RETI ; AND RETURN FROM INTERRUPT ROUTINE
		150	
		151	END







Based on

8051

October 1984

...with only 4 1/2" displays.

- 1) CMT Terminal Basics  
2) 8051 Description  
3) 8255 Description  
4) Design Backgroun  
5) System Description

01248 JAHMREY THO 0.5

roller Terminal

**Michael A. Shafer**  
Microcontroller Applications

# 8051 Based CRT Terminal Controller

**Michael A. Shafer**  
Microcontroller Applications

## 1.0 INTRODUCTION

This is the third application note that Intel has produced on CRT terminal controllers. The first Ap Note (ref. 1), written in 1977, used the 8080 as the CPU and required 41 packages including 11 LSI devices. In 1979, another application note (ref. 2) using the 8085 as the controller was produced and the chip count decreased to 20 with 11 LSI devices.

Advancing technology has integrated a complete system onto a single device that contains a CPU, program memory, data memory, serial communication, interrupt controller, and I/O. These "computer-on-a-chip" devices are known as microcontrollers. Intel's MCS<sup>®</sup>-51 microcontroller was chosen for this application because of its highly integrated functions. This CRT terminal design uses 12 packages with only 4 LSI devices.

This application note has been divided into five general sections:

- 1) CRT Terminal Basics
- 2) 8051 Description
- 3) 8276 Description
- 4) Design Background
- 5) System Description

## 2.0 CRT TERMINAL BASICS

A terminal provides a means for humans to communicate with a computer. Terminals may be as simple as a LED display and a couple of push buttons, or it may be an elaborate graphics system that contains a full function keyboard with user programmable keys, color CRT and several processors controlling its functions. This application note describes a basic low cost terminal containing a black and white CRT display, full function keyboard and a serial interface.

### 2.1 CRT Description

A raster scan CRT displays its images by generating a series of lines (raster) across the face of the tube. The electron beam usually starts at the top left hand corner moves left to right, back to the left of the screen, moves down one row and continues on to the right. This is repeated until the lower right hand of the screen is reached. Then the beam returns to the top left hand corner and refreshes the screen. The beam forms a zigzag pattern as shown in Figure 2.1.0.

Two independent operating circuits control this movement across the screen. The horizontal oscillator controls the left to right motion of the beam while the vertical controls the top to bottom movement. The vertical oscillator also tells the beam when to return to the upper left hand corner or "home" position.

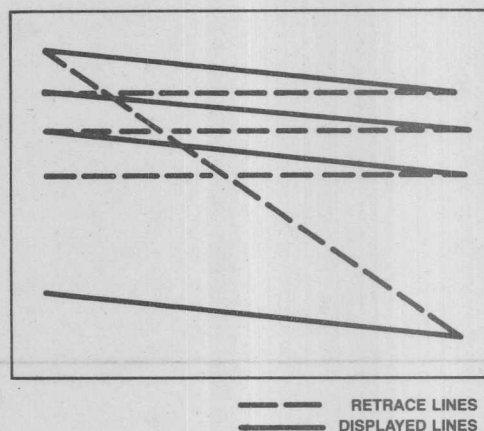


Figure 2.1.0 Raster Scan

As the electron beam moves across the screen under the control of the horizontal oscillator, a third circuit controls the current entering the electron gun. By varying the current, the image may be made as bright or as dim as the user desires. This control is also used to turn the beam off or "blank the screen".

When the beam reaches the right hand side of the screen, the beam is blanked so it does not appear on the screen as it returns to the left side. This "retrace" of the beam is at a much faster rate than it traveled across the screen to generate the image.

The time it takes to scan the whole screen and return to the home position is referred to as a "frame". In the United States, commercial television broadcast uses a horizontal sweep frequency of 15,750Hz which calculates out to 63.5 microseconds per line. The frame time is equal to 16.67 milliseconds or 60Hz vertical sweep frequency.

Although this is the commercial standard, many CRT displays operate from 18KHz to 30KHz horizontal frequency. As the horizontal frequency increases, the number of lines per frame increases. This increase in lines or resolution is needed for graphic displays and on special text editors that display many more lines of text than the standard 24 or 25 character lines.

Since the United States operates on a 60Hz A.C. power line frequency, most CRT monitors use 60Hz as the vertical frequency. The use of 60Hz as the vertical frequency allows the magnetic and electrical variations that can modulate the electron beam to be synchronized with the display, thus they go unnoticed. If a frequency other than 60Hz is used, special shielding and power supply regu-

lating is usually required. Very few CRTs operate on a vertical frequency other than 60Hz due to the increase in the overall system cost.

The CRT controller must generate the pulses that define the horizontal and vertical timings. On most raster scan CRTs the horizontal frequency may vary as much as 500Hz without any noticeable effect on the quality of the display. This variation can change the number of horizontal lines from 256 to 270 per frame.

The CRT controller must also shift out the information to be displayed serially to the circuit that controls the electron beam's intensity as it scans across the screen. The circuits that control the timing associated with the shifting of the information are known as the dot clock and the character clock. The character clock frequency is equal to the dot clock frequency divided by the number of dots it takes to form a character in the horizontal axis. The dot clock frequency is calculated by the following equation:

$$\text{Dot Clock (Hz)} = (N + R) * D * L * F$$

where

- N is the number of displayed characters per row,
- R is the number of character times for the retrace,
- D is the number of dots per character in the horizontal axis,
- L is the number of horizontal lines per frame,
- F is the frame rate in Hz.

In this design N = 80, R = 20, D = 7, L = 270, and F = 60Hz. Plugging in the numbers results in a dot clock frequency of 11.34MHz.

The retrace number may vary on each design because it is used to set the left and right hand margins on the CRT. The number of dots per character is chosen by the designer to meet the system needs. In this design, a 5 x 7 dot matrix and 2 blank dots between each character (see Figure 2.1.1) makes D equal to 5 + 2 = 7.

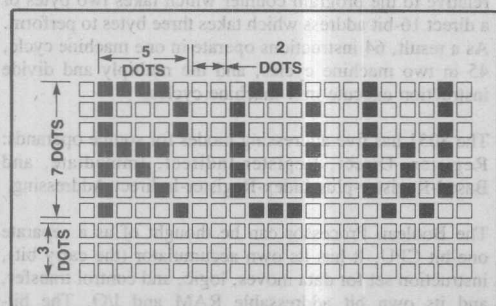


Figure 2.1.1 5 x 7 Dot Matrix

The following equation can be used to figure the number of lines per frame:

$$L = (H * Z) + V$$

where

- H is the number of horizontal lines per character,
- Z is the number of character lines per frame,
- V is the number of horizontal line times during the vertical retrace

In this design H is equal to the 7 horizontal dots per character plus 3 blank dots between each row which adds up to 10. Also 25 lines of characters are displayed, so Z = 25. The vertical retrace time is variable to set the top and bottom margins on the CRT and in this design is equal to 20. Plugging in the numbers gives L = 270 lines per frame.

## 2.2 Keyboard

A keyboard is the common way a human enters commands and data to a computer. A keyboard consists of a matrix of switches that are scanned every couple of milliseconds by a keyboard controller to determine if one of the keys has been pressed. Since the keyboard is made up of mechanical switches that tend to bounce or "make and break" contact everytime they are pressed, debouncing of the switches must also be a function of the keyboard controller. There are dedicated keyboard controllers available that do everything from scanning the keyboard, debouncing the keys, decoding the ASCII code for that key closure to flagging the CPU that a valid key has been depressed. The keyboard controller may present the information to the CPU in parallel form or in a serial data stream.

This Application Note integrates the function of the keyboard controller into the 8051 which is also the terminal controller. Provisions have been made to interface the 8051 to a keyboard that uses a dedicated keyboard controller. The 8051 can accept data from the keyboard controller in either parallel or serial format.

## 2.3 Serial Communications

Communication between a host computer and the CRT terminal can be in either parallel or serial data format. Parallel data transmission is needed in high end graphic terminals where great amounts of information must be transferred.

One can rarely type faster than 120 words per minute, which corresponds to 12 characters per second or 1 character per 83 milliseconds. The utilization of a parallel port cannot justify the cost associated with the drivers and the amount of wire needed to perform this transmission. Full duplex serial data transmission requires 3 wires and two



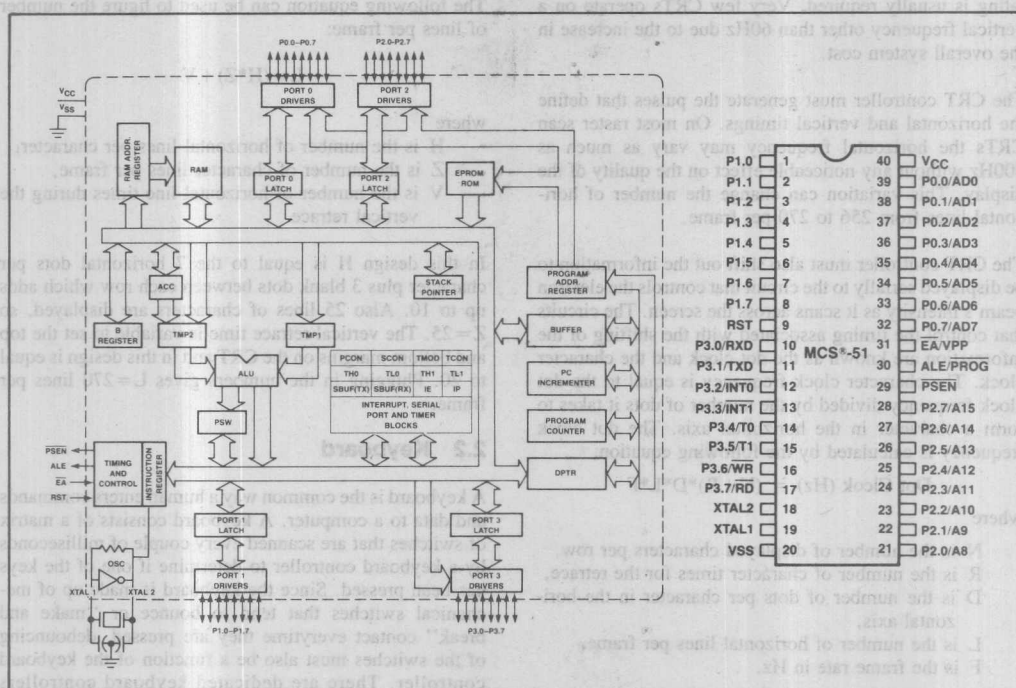


Figure 3.0.0 8051 Block Diagram

drivers to implement the communication channel between the host computer and the terminal. The data rate can be as high as 19200 BAUD in the asynchronous serial format. BAUD rate is the number of bits per second received or transmitted. In the asynchronous serial format, 10 bits of information is required to transmit one character. One character per 500 microseconds or 1,920 characters per second would then be transmitted using 19.2 KBAUD.

This application note uses the 8051 serial port configured for full duplex asynchronous serial data transmission. The software for the 8051 has been written to support variable BAUD rates from 150 BAUD up to 9.6 KBAUD.

### 3.0 8051 DESCRIPTION

The 8051 is a single chip high-performance microcontroller. A block diagram is shown in figure 3.0.0. The 8051 combines CPU; Boolean processor; 4K  $\times$  8 ROM; 128  $\times$  8 RAM; 32 I/O lines; two 16-bit timer/ event counters; a five-source, two-priority-level, nested interrupt structure; serial I/O port for either multiprocessor communications, I/O expansion, or full duplex UART; and on-chip oscillator and clock circuits.

### 3.1 CPU

Efficient use of program memory results from an instruction set consisting of 49 single-byte, 45 two-byte and 17 three-byte instructions. Most arithmetic, logical and branching operations can be performed using an instruction that appends either a short address or a long address. For example, branches may use either an offset that is relative to the program counter which takes two bytes or a direct 16-bit address which takes three bytes to perform. As a result, 64 instructions operate in one machine cycle, 45 in two machine cycles, and the multiply and divide instruction execute in 4 machine cycles.

The 8051 has five addressing modes for source operands: Register, Direct, Register-Indirect, Immediate, and Based-Register-plus Index-Register-Indirect Addressing.

The Boolean Processor can be thought of as a separate one-bit CPU. It has its own accumulator (the carry bit), instruction set for data moves, logic, and control transfer, and its own bit addressable RAM and I/O. The bit-manipulating instructions provide optimum code and speed efficiency for handling on chip peripherals. The

Boolean processor also provides a straight forward means of converting logic equations directly into software. Complex combinational logic functions can be resolved without extensive data movement, byte masking, and test-and-branch trees.

### 3.2 On-Chip Ram

The CPU manipulates operands in four memory spaces. These are the 64K-byte Program Memory, 64K-byte External Data Memory, 128-byte Internal Data Memory, and 128-byte Special Function Registers (SFRs). Four Register Banks (each with 8 registers), 128 addressable bits, and the Stack reside in the internal Data RAM. The Stack size is limited only by the available Internal Data RAM and its location is determined by the 8-bit Stack Pointer. All registers except for the Program Counter and the four 8-Register Banks reside in the SFR address space. These memory mapped registers include arithmetic registers, pointers, I/O ports, and registers for the interrupt system, timers, and serial channel.

Registers in the four 8-Register Banks can be addressed by Register, Direct, or Register-Indirect Addressing modes. The 128 bytes of internal Data Memory can be addressed by Direct or Register-Indirect modes while the SFRs are only addressed directly.

### 3.3 I/O Ports

The 8051 has instructions that can treat the 32 I/O lines as 32 individually addressable bits or as 4 parallel 8-bit ports addressable as Ports 0, 1, 2, and 3.

Resetting the 8051 writes a logical 1 to each pin on port 0 which places the output drivers into a high-impedance mode. Writing a logical 0 to a pin forces the pin to ground and sinks current. Re-writing the pin high will place the pin in either an open drain output or high-impedance input mode.

Ports 1, 2, and 3 are known as quasi-bidirectional I/O pins. Resetting the device writes a logical one to each pin. Writing a logical 0 to the pin will force the pin to ground and sink current. Re-writing the pin high will place the pin in an output mode with a weak depletion pullup FET or in the input mode. The weak pullup FET is easily overcome by a TTL output.

Ports 0 and 2 can also be used for off-chip peripheral expansion. Port 0 provides a multiplexed low-order address and data bus while Port 2 contains the high-order address when using external Program Memory or more than 256 byte external Data Memory.

Port 3 pins can also be used to provide external interrupt request inputs, event counter inputs, the serial port TXD

and RXD pins and to generate control signals used for writing and reading external peripherals.

### 3.4 Interrupt System

External events and the real-time-driven on-chip peripherals require service by the CPU asynchronous to the execution of any particular section of code. A five-source, two-level, nested interrupt system ties the real time events to the normal program execution.

The 8051 has two external interrupt sources, one interrupt from each of the two timer/counters, and an interrupt from the serial port. Each interrupt vectors the program execution to its own unique memory location for servicing the interrupt. In addition, each of the five sources can be individually enabled or disabled as well as assigned to one of the two interrupt priority levels available on the 8051.

Up to two additional external interrupts can be created by configuring a timer/counter to the event counter mode. In this mode the timer/counter increments on command by either the T0 or T1 pin. An interrupt is generated when the timer/counter overflows. Thus if the timer/counter is loaded with the maximum count, the next high-to-low transition of the event counter input will cause an interrupt to be generated.

### 3.5 Serial Port

The 8051's serial port is useful for linking peripheral devices as well as multiple 8051s through standard asynchronous protocols with full duplex operation. The serial port also has a synchronous mode for expansion of I/O lines using shift registers. This hardware serial port saves ROM code and permits a much higher transmission rate than could be achieved through software. The processor merely needs to read or write the serial buffer in response to an interrupt. The receiver is double buffered to eliminate the possibility of overrun if the processor failed to read the buffer before the beginning of the next frame.

The full duplex asynchronous serial port provides the means of communication with standard UART devices such as CRT terminals and printers.

The reader should refer to the microcontroller handbook for a complete discussion of the 8051 and its various modes of operation.

## 4.0 8276 DESCRIPTION

The 8276's block diagram and pin configuration are shown in Figure 4.0.0. The following sections describe the general capabilities of the 8276.

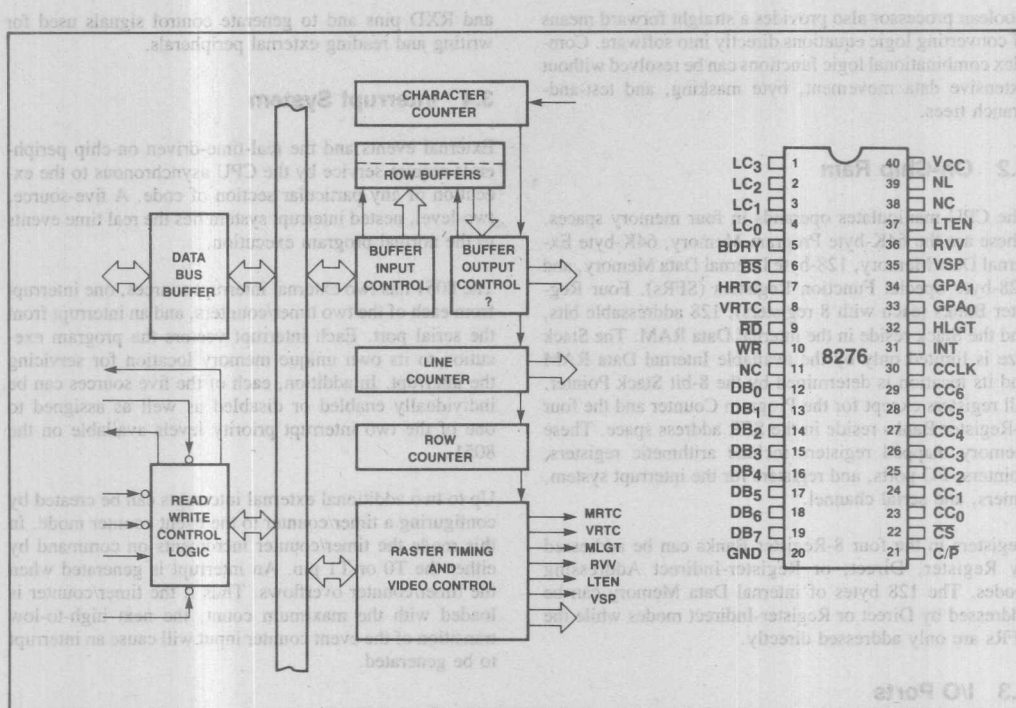


Figure 4.0.0 8276 Block Diagram

#### 4.1 CRT Display Refreshing

The 8276, having been programmed by the system designer for a specific screen format, generates a series of Buffer Ready signals. A row of characters is then transferred by the system controller from the display memory to the 8276's row buffers. The row buffers are filled by deselecting the 8276 CS and asserting the BS and WR signals. The 8276 presents the character codes to an external character generator ROM by using outputs CC0-CC6. The parallel data from the outputs of the character generator is converted to serial information that is clocked by external dot timing logic into the video input of the CRT.

The character rows are displayed on the CRT one line at a time. Line count outputs LC0-LC3 select the current line information from the character generator ROM. The display process is illustrated in Figure 4.1.0. This process is repeated for each display character row. At the beginning of the last display row the 8276 generates an interrupt request by raising its INT output line. The interrupt request

is used by the 8051 system controller to reinitialize its load buffer pointers for the next display refresh cycle.

Proper CRT refreshing requires that certain 8276 parameters be programmed at system initialization time. The 8276 has two types of internal registers; the write only Command (CREG) and Parameter (PREG) Registers, and the read only Status Register (SREG). The 8276 expects to receive a command followed by 0 to 4 parameter bytes depending on the command. A summary of the 8276's instruction set is shown in Figure 4.1.1. To access the registers, CS must be asserted along with WR or RD. The status of the C/P pin determines whether the command or parameter registers are selected.

The 8276 allows the designer flexibility in the display format. The display may be from 1 to 80 characters per row, 1 to 64 rows per screen, and 1 to 16 horizontal lines per character row. In addition, four cursor formats are available; blinking, non-blinking, underline, and reverse video. The cursor position is programmable to anywhere on the screen via the Load Cursor command.

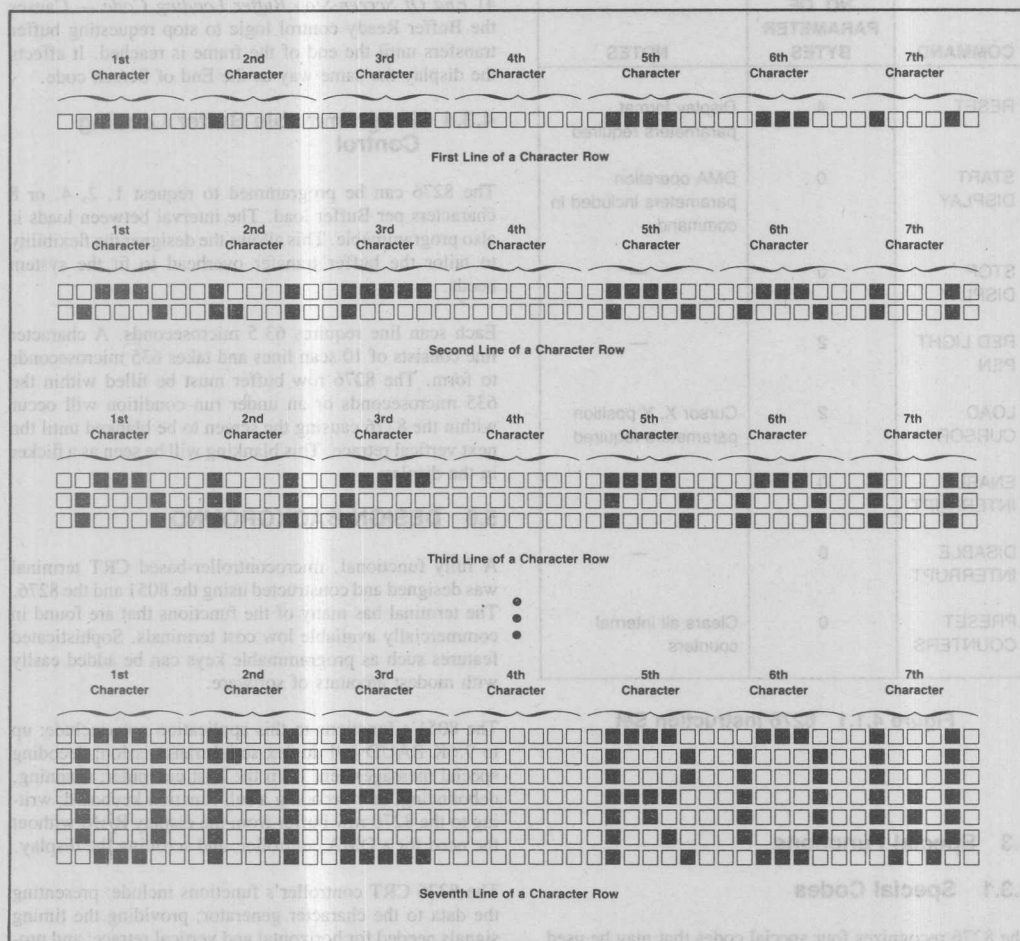


Figure 4.1.0 8276 Row Display

## 4.2 CRT Timing

The 8276 provides two timing outputs for controlling the CRT. The Horizontal Retrace Timing and Control (HRTC) and Vertical Retrace Timing and Control (VRTC) signals are used for synchronizing the CRT horizontal and vertical oscillators. A third output, VSP (Video Suppress), provides a signal to the dot timing logic to blank the video signal during the horizontal and vertical retraces. LTEN (Light Enable) is used to provide the ability to force the

video output high regardless of the state of the VSP signal. This feature is used to place the cursor on the screen and to control attribute functions.

RVV (Reverse Video) output, if enabled, will cause the system to invert its video output. The fifth timing signal output, HLGT (highlight) allows the flexibility to increase the CRT beam intensity to a greater than normal level.



COMMAND	NO. OF PARAMETER BYTES	NOTES
RESET	4	Display format parameters required
START DISPLAY	0	DMA operation parameters included in command
STOP DISPLAY	0	—
RED LIGHT PEN	2	—
LOAD CURSOR	2	Cursor X, Y position parameters required
ENABLE INTERRUPT	0	—
DISABLE INTERRUPT	0	—
PRESET COUNTERS	0	Clears all internal counters

Figure 4.1.1 8276 Instruction Set

### 4.3 Special Functions

#### 4.3.1 Special Codes

The 8276 recognizes four special codes that may be used to reduce memory, software, or system controller overhead. These characters are placed within the display memory by the system controller. The 8276 performs certain tasks when these codes are received in its row buffer memory.

- 1) *End of Row Code* — Activates VSP. VSP remains active until the end of the line is reached. While VSP is active the screen is blanked.
- 2) *End Of Row-Stop Buffer Loading Code* — Causes the Buffer Ready control logic to stop requesting buffer transfers for the rest of the row. It affects the display the same as End of Row Code.
- 3) *End Of Screen Code* — Activates VSP. VSP remains active until the end of the frame is reached.

4) *End Of Screen-Stop Buffer Loading Code* — Causes the Buffer Ready control logic to stop requesting buffer transfers until the end of the frame is reached. It affects the display the same way as the End of Screen code.

#### 4.3.4 Programmable Buffer Loading Control

The 8276 can be programmed to request 1, 2, 4, or 8 characters per Buffer load. The interval between loads is also programmable. This allows the designer the flexibility to tailor the buffer transfer overhead to fit the system needs.

Each scan line requires 63.5 microseconds. A character line consists of 10 scan lines and takes 635 microseconds to form. The 8276 row buffer must be filled within the 635 microseconds or an under run condition will occur within the 8276 causing the screen to be blanked until the next vertical retrace. This blanking will be seen as a flicker in the display.

### 5.0 DESIGN BACKGROUND

A fully functional, microcontroller-based CRT terminal was designed and constructed using the 8051 and the 8276. The terminal has many of the functions that are found in commercially available low cost terminals. Sophisticated features such as programmable keys can be added easily with modest amounts of software.

The 8051's functions in this application note include: up to 9.6K BAUD full duplex serial transmission; decoding special messages sent from the host computer; scanning, debouncing, and decoding a full function keyboard; writing to the 8276 row buffer from the display RAM without the need for a DMA controller; and scrolling the display.

The 8276 CRT controller's functions include: presenting the data to the character generator; providing the timing signals needed for horizontal and vertical retrace; and providing blanking and video information.

#### 5.1 Design Philosophy

Since the device count relates to costs, size, and reliability of a system, arriving at a minimum device count without degrading the performance was a driving force for this application note. LSI devices were used where possible to maintain a low chip count and to make the design cycle as short as possible.

PL/M-51 was chosen to generate the majority of the software for this application because it models the human thought process more closely than assembly language. Consequently it is easier and faster to write programs using PL/M-51 and the code is more likely to be correct because less chance exists to introduce errors.

PL/M-51 programs are easier to read and follow than assembly language programs, and thus are easier to modify and customize to the end user's application. PL/M-51 also offers lower development and maintenance costs than assembly language programming.

PL/M-51 does have a few drawbacks. It is not as efficient in code generation relative to assembly language and thus may also run slower.

This application note uses the 8051's interrupts to control the servicing of the various peripherals. The speed of the main program is less critical if interrupts are used. In the last two application notes on terminal controllers, a criterion of the system was the time required for receiving an incoming serial byte, decoding it, performing the function requested, scanning the keyboard, debouncing the keys, and transmitting the decoded ASCII code must be less than the vertical refresh time. Using the 8051 and its interrupts makes this time constraint irrelevant.

## 5.2 System Target Specifications

The design specifications for the CRT terminal design is as follows:

### Display Format

- 80 characters/display row
- 25 display lines

### Character Format

- $51 \times 7$  character contained within a  $7 \times 10$  frame
- First and seventh columns blanked
- Ninth line cursor position
- Programmable delay blinking underline cursor

### Control Characters Recognized

- Backspace
- Linefeed
- Carriage Return
- Form Feed

### Escape Sequences Recognized

- ESC A, Cursor up
- ESC B, Cursor down
- ESC C, Cursor right
- ESC D, Cursor left
- ESC E, Clear screen
- ESC F, Move addressable cursor
- ESC H, Home cursor
- ESC J, Erase from cursor to the end the screen
- ESC K, Erase the current line

### Characters Displayed

- 96 ASCII Alphanumeric Characters

### Characters Transmitted

- 96 ASCII Alphanumeric Characters
- ASCII Control Character Set
- ASCII Escape Sequence Set
- Auto Repeat

### Display Memory

- $2K \times 8$  static RAM

### Data Rate

- Variable rate from 150 to 9600 BAUD

### CRT Monitor

- Ball Bros TV-12, 12MHZ Black and White

### Keyboard

- Any standard undecoded keyboard (2 key lock-out)
- Any standard decoded keyboard with output enable pin
- Any standard decoded serial keyboard up to 150 BAUD

### Scrolling Capability

### Compatible With Wordstar

## 6.0 SYSTEM DESCRIPTION

A block diagram of the CRT terminal is shown in figure 6.0.0. The diagram shows only the essential system features. A detailed schematic of the CRT terminal is contained in the Appendix 7.1.

The "brains" of the CRT terminal is the 8051 microcontroller. The 8276 is the CRT controller in the system, and a 2716 EPROM is used as the character generator. To handle the high speed portion of the CRT, the 8276 is surrounded by a handful of TTL devices. A  $2K \times 8$  static RAM was used as the display memory.

Following the system reset, the 8276 is initialized for cursor type, number of characters per line, number of lines, and character size. The display RAM is initialized to all "spaces" (ASCII 20H). The 8051 then writes the "start display" command to the 8276. The local/line input is sampled to determine the terminal mode. If the terminal is on-line, the BAUD rate switches are read and the serial port is set up for full duplex UART mode. The processor then is put into a loop waiting to service the serial port fifo or the 8276.

The serial port is programmed to have the highest priority interrupt. If the serial port generates an interrupt, the processor reads the buffer, puts the character in a generated fifo that resides in the 8051's internal RAM, increments the fifo pointer, sets the serial interrupt flag and returns.

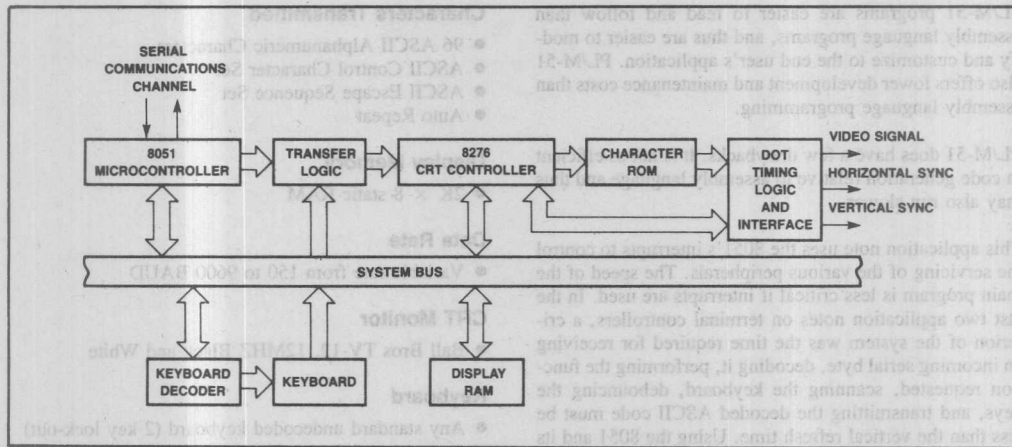


Figure 6.0.0 CRT Terminal Controller Block Diagram

The main program determines if it is a displayable character, a Control word or an ESC sequence and either puts the character in the display buffer or executes the appropriate command sent from the host computer.

If the 8276 needs servicing, the 8051 fills the row buffer for the CRT display's next line. If the 8276 generates a vertical retrace interrupt, the buffer pointers are reloaded with the display memory location that corresponds to the first character of the first display line on the CRT. The vertical retrace also signals the processor to read the keyboard for a key closure.

## 6.1 Hardware Description

The following section describes the unique characteristics of this design.

### 6.1.1 Peripheral Address Map

The display RAM, 8276 registers, and the 8276 row buffers are memory mapped into the external data RAM address area. The addresses are as follows:

Read and Write External Display RAM —	Address 1000H to 17CFH
Write to 8276 row buffers from Display RAM —	Address 1800H to 1FCFH
Write to 8276 Command Register (CREG) —	Address 0001H
Write to 8276 Parameter Register (PREG) —	Address 0000H
Read from 8276 Status Register (SREG) —	Address 0001H

Three general cases can be explored; reading and writing the display RAM, writing to the 8276 row buffers, and reading and writing the 8276's control registers.

As mentioned previously the 8051 fills the 8276 row buffer without the need of a DMA controller. This is accomplished by using a Quad 2-input multiplexor (Figure 6.1.0) as the transfer logic shown in the block diagram. The address line, P2.3, is used to select either of the two inputs. When the address line is low the  $\overline{RD}$  and  $\overline{WR}$  lines perform their normal functions, that is read and write the

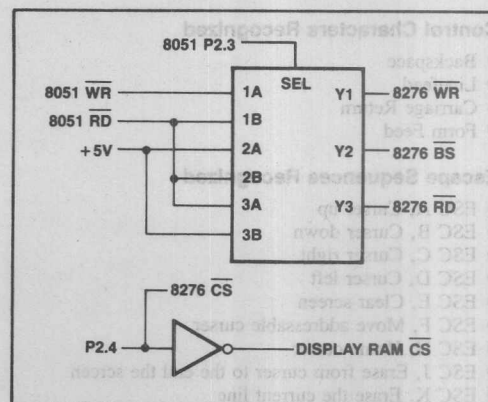


Figure 6.1.0 Simplified Version Of The Transfer Logic

8276 or the external display RAM depending on the states of their respective chip selects. If the address line is high, the 8051 RD line is transformed into BS and WR signals for the 8276. While holding the address line high, the 8051 executes an external data move (MOVX) from the display RAM to the accumulator which causes the display RAM to output the addressed byte onto the data bus. Since the multiplexor turns the same 8051 RD pulses into BS and WR pulses to the 8276, the data bus is thus read into the 8276 as a Buffer transfer. This scheme allows 80 characters to be transferred from the display RAM into the 8276 within the required character line time of 635 microseconds. The 8051 easily meets this requirement by accomplishing the task within 350 microseconds.

### 6.1.2 Scanning The Keyboard

Throughout this project, provision have been made to make the overall system flexible. The software has been written for various keyboards and the user simply needs to link different program modules together to suit their needs.

#### 6.1.2.1 Undecoded Keyboard

Incorporating an undecoded keyboard controller into the other functions of the 8051 shows the flexibility and over all CPU power that is available. The keyboard in this case is a full function, non-buffered  $8 \times 8$  matrix of switches for a total of 64 possible keys. The 8 send lines are connected to a 3-to-8 open-collector decoder as shown in Figure 6.1.1. Three high order address lines from the 8051 are the decoder inputs. The enabling of the decoder is accomplished through the use of the PSEN signal from the 8051 which makes the architecture of the separate address space for the program memory and the external data RAM work for us to eliminate the need to decode addresses externally. The move code (MOVC) instruction allows each scan line of the keyboard to be read with one instruction.

The keyboard is read by bringing one of the eight scan lines low sequentially while reading the return lines which are pulled high by an external resistor. If a switch is

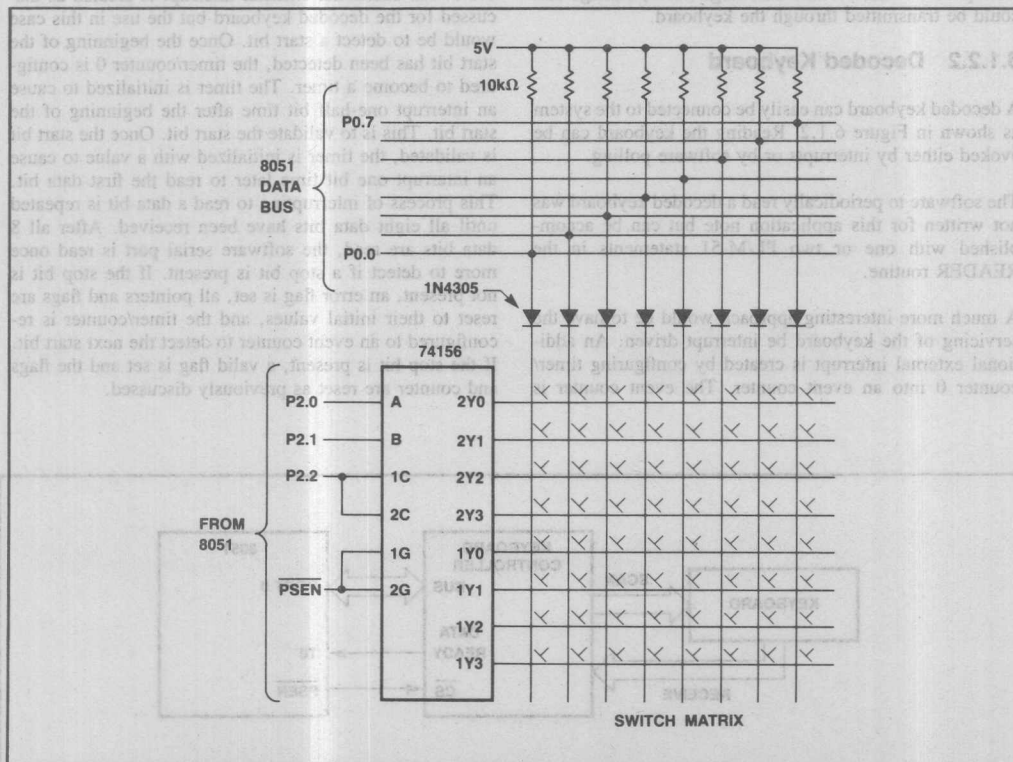


Figure 6.1.1 Keyboard



closed, the data bus line is connected through the switch to the low output of the decoder and one of the data bus lines will be read as a 0. By knowing which scan line detected a key closure and which data bus line was low, the ASCII code for that key can easily be looked up in a matrix of constants. PL/M-51 has the ability to handle arrays and structured arrays, which makes the decoding of the keyboard a trivial task.

Since the Shift, Cap Lock, and Control keys may change the ASCII code for a particular key closure, it is essential to know the status of these pins while decoding the keyboard. The Shift, Cap Lock, and Control keys are therefore not scanned but are connected to the 8051 port pins where they can be tested for closure directly.

The 8 receive lines are connected to the data bus through germanium diodes which chosen for their low forward voltage drop. The diodes keep the keyboard from interfering with the data bus during the times the keyboard is not being read. The circuit consisting of the 3-to-8 decoder and the diodes also offers some protection to the 8051 from possible Electrostatic Discharge (ESD) damage that could be transmitted through the keyboard.

### 6.1.2.2 Decoded Keyboard

A decoded keyboard can easily be connected to the system as shown in Figure 6.1.2. Reading the keyboard can be evoked either by interrupts or by software polling.

The software to periodically read a decoded keyboard was not written for this application note but can be accomplished with one or two PL/M-51 statements in the READER routine.

A much more interesting approach would be to have the servicing of the keyboard be interrupt driven. An additional external interrupt is created by configuring timer/counter 0 into an event counter. The event counter is

initialized with the maximum count. The keyboard controller would inform the 8051 that a valid key has been depressed by pulling the input pin T0 low. This would overflow the event counter, thus causing an interrupt. The interrupt routine would simply use a MOVX (PSEN is connected to the output enable pin of the keyboard controller) to read the contents of the keyboard controller onto the data bus, reinitialize the counter to the maximum count and return from the interrupt.

### 6.1.2.3 Serial Decoded Keyboard

The use of detachable keyboards has become popular among the manufacturers of keyboards and personal computers. This terminal has provisions to use such a keyboard.

The keyboard controller would scan the keyboard, debounce the key and send back the ASCII code for that key closure. The message would be in an asynchronous serial format.

The flowchart for a software serial port is shown in Figure 6.1.3. An additional external interrupt is created as discussed for the decoded keyboard but the use in this case would be to detect a start bit. Once the beginning of the start bit has been detected, the timer/counter 0 is configured to become a timer. The timer is initialized to cause an interrupt one-half bit time after the beginning of the start bit. This is to validate the start bit. Once the start bit is validated, the timer is initialized with a value to cause an interrupt one bit time later to read the first data bit. This process of interrupting to read a data bit is repeated until all eight data bits have been received. After all 8 data bits are read, the software serial port is read once more to detect if a stop bit is present. If the stop bit is not present, an error flag is set, all pointers and flags are reset to their initial values, and the timer/counter is re-configured to an event counter to detect the next start bit. If the stop bit is present, a valid flag is set and the flags and counter are reset as previously discussed.

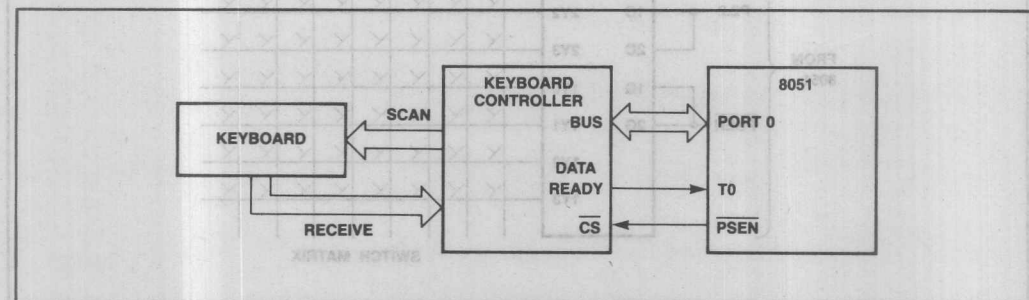


Figure 6.1.2 Using A Decoded Keyboard

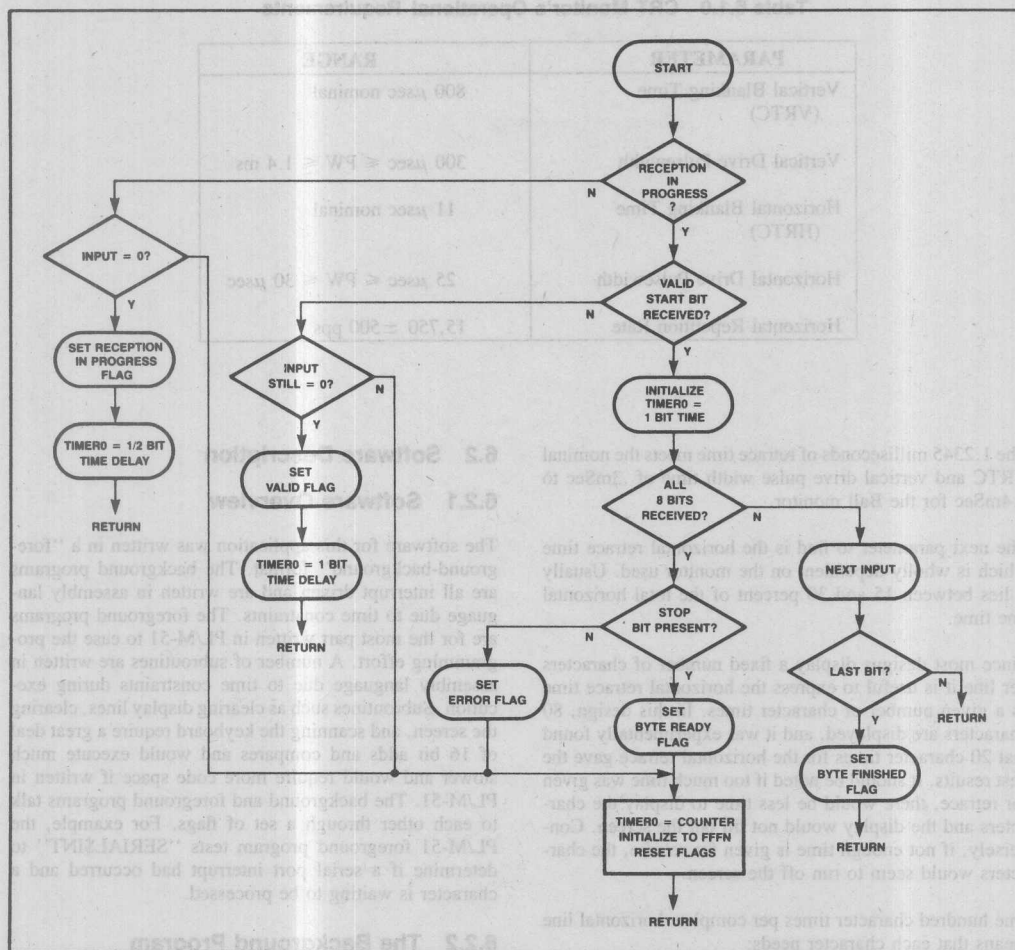


Figure 6.1.3 Flowchart for the Software Serial Port

### 6.1.4 System Timings

The requirements for the BALL BROTHERS. TV-12 monitor's operation is shown in table 6.1.0. From the monitor's parameters, the 8276 specifications and the system target specifications the system timing is easily calculated.

The 8276 allows the vertical retrace to be only an integer multiple of the horizontal character lines. Twenty-five display lines and a character frame of  $7 \times 10$  are required from the target specification which will require 250 horizontal lines. If the horizontal frequency is to be within

500 Hz of 15,750 Hz, we must choose either one or two character line times for horizontal retrace. To allow for a little more margin at the top and bottom of the screen, two character line times was chosen for the vertical retrace. This choice yields  $250 + 20 = 270$  total character lines per frame. Assuming 60 Hz vertical retrace frequency:

$60 \text{ Hz} \times 270 = 16,200 \text{ Hz horizontal frequency}$   
and  
 $1/16,200 \text{ Hz} \times 20 \text{ horizontal sync times} = 1.2345 \text{ milliseconds}$

Table 6.1.0 CRT Monitor's Operational Requirements

PARAMETER	RANGE
Vertical Blanking Time (VRTC)	800 $\mu$ sec nominal
Vertical Drive Pulsewidth	300 $\mu$ sec $\leq$ PW $\leq$ 1.4 ms
Horizontal Blanking Time (HRTC)	11 $\mu$ sec nominal
Horizontal Drive Pulsewidth	25 $\mu$ sec $\leq$ PW $\leq$ 30 $\mu$ sec
Horizontal Repetition Rate	15,750 $\pm$ 500 pps

The 1.2345 milliseconds of retrace time meets the nominal VRTC and vertical drive pulse width time of .3mSec to 1.4mSec for the Ball monitor.

The next parameter to find is the horizontal retrace time which is wholly dependent on the monitor used. Usually it lies between 15 and 30 percent of the total horizontal line time.

Since most designs display a fixed number of characters per line it is useful to express the horizontal retrace time as a given number of character times. In this design, 80 characters are displayed, and it was experimentally found that 20 character times for the horizontal retrace gave the best results. It should be noted if too much time was given for retrace, there would be less time to display the characters and the display would not fill out the screen. Conversely, if not enough time is given for retrace, the characters would seem to run off the screen.

One hundred character times per complete horizontal line means that each character needs:

$$(1/16,200 \text{ Hz}) / 100 \text{ character times} = 617.3 \text{ nanoseconds}$$

If we multiply the 20 character times needed to retrace by 617.3 nanoseconds needed for each character, we find 12.345 microseconds are allocated for retrace. This value falls short of the 25 to 30 microseconds required by the horizontal drive of the Ball monitor. To correct for this, a 74LS123 one-shot was used to extend the horizontal drive pulse width.

The dot clock frequency is easy to calculate now that we know the horizontal frequency. Since each character is formed by seven dots in the horizontal axis, the dot clock period would be the character clock (617.3 nanoseconds) divided by the 7 which is equal to 11.34 MHz. The basic dot timing and CRT timing are shown in the Appendix.

## 6.2 Software Description

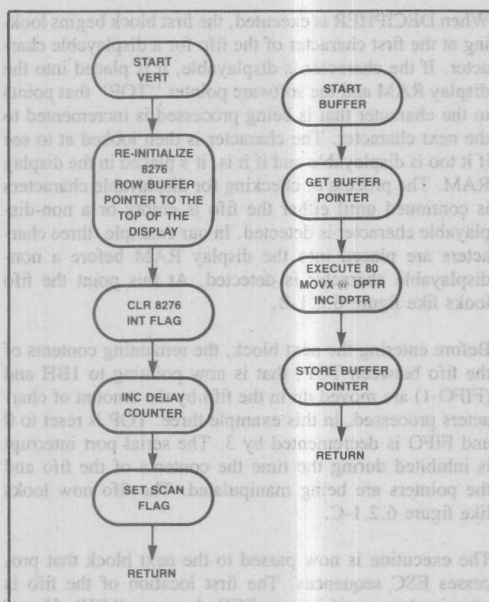
### 6.2.1 Software Overview

The software for this application was written in a "foreground-background" format. The background programs are all interrupt driven and are written in assembly language due to time constraints. The foreground programs are for the most part written in PL/M-51 to ease the programming effort. A number of subroutines are written in assembly language due to time constraints during execution. Subroutines such as clearing display lines, clearing the screen, and scanning the keyboard require a great deal of 16 bit adds and compares and would execute much slower and would require more code space if written in PL/M-51. The background and foreground programs talk to each other through a set of flags. For example, the PL/M-51 foreground program tests "SERIAL\$INT" to determine if a serial port interrupt had occurred and a character is waiting to be processed.

### 6.2.2 The Background Program

Two interrupt driven routines, VERT and BUFFER, (see Fig. 6.2.0) request service every 16.67 milliseconds and 617 microseconds respectively. VERT's request comes during the last character row of the display screen. This routine resets the buffer pointers to the first CRT display line in the display memory. VERT is also used as a time base for the foreground program. VERT sets the flag, SCAN, to tell the foreground program (PL/M-51) that it is time to scan the Keyboard. VERT also increments a counter used for the delay between transmitting characters in the AUTO\$REPEAT routine.

The BUFFER routine is executed once per character row. BUFFER uses the multiplexor discussed earlier to fill the 8276's row buffer by executing 80 external data moves and incrementing the Data Pointer between each move.



**Figure 6.2.0 Flowcharts For VERT and BUFFER Routine**

The MOVX reads the display RAM and writes the character into the row buffer during the same instruction.

SERBUF is an interrupt driven routine that is executed each time a character is received or transmitted through the on-chip serial port. The routine first checks if the interrupt was caused by the transmit side of the serial port, signaling that the transmitter is ready to accept another character. If the transmitter caused the interrupt, the flag "TRANSMITSINT" is set which is checked by the foreground program before putting a character in the buffer for transmission.

If the receiver caused the interrupt, the input buffer on the serial port is read and fed into the fifo that has been manufactured in the internal RAM and increments the fifo pointer "FIFO." The flag "SERIALSINT" is then set, telling the foreground program that there is a character in the fifo to be processed. If the read character is an ESC character, the flag "ESCSEQ" is set to tell the foreground program that an escape sequence is in the process of being received.

### 6.2.3 The Foreground Program

The foreground program is documented in the Appendix. The foreground program starts off by initializing the 8276

as discussed earlier. After all variables and flags are initialized, the processor is put into a loop waiting for either VERT to set SCAN so the program can scan the keyboard, or for the serial port to set SERIALSINT so the program can process the incoming character.

The vertical retrace is used to time the delay between keyboard scans. When VERT gets set, the assembly language routine READER is called. READER scans the keyboard, writing each scan into RAM to be processed later. READER controls two flags, KEY0 and SAME. KEY0 is set when all 8 scans determine that no key is pressed. SAME is set when the same key that was pressed last time the keyboard was read is still pressed.

After READER returns execution to the main program, the flags are tested. If the KEY0 flag is set the main program goes back to the loop waiting for the vertical retrace or a serial port interrupt to occur. If the SAME flag is set the main program knows that the closed key has been debounced and decoded so it sends the already known ASCII code to the AUTOREPEAT routine which determines if that character should be transmitted or not.

If KEY0 and SAME are not set, signifying that a key is pressed but it is not the same key as before, the foreground program determines if the results from the scan are valid. First all eight scans are checked to see if only one key was closed. If only one key is closed, the ASCII code is determined, modified if necessary by the Shift, Cap Lock, or Control keys. The NEWSKEY and VALID flags are then set. The next time READER is called, if the same key is still pressed, the SAME flag will be set, causing the AUTOREPEAT subroutine to be called as just discussed. Since the keyboard is read during the vertical retrace, 16.67 milliseconds has elapsed between the detection of the pressed key and reverifying that the key is still pressed before transmitting it, thus effectively debouncing the key.

The AUTOREPEAT routine is written to transmit any key that the NEWSKEY flag is set for. The counter that is incremented each time the vertical refresh interrupt is serviced causes a programmable delay between the first transmission and subsequent auto repeat transmission. Once the NEWSKEY character is sent, the counter is initialized. Each time the AUTOREPEAT routine is called, the counter is checked. Only when the counter overflows will the next character be transmitted. After the initial delay, a character will be transmitted every other time the routine is called as long as the key remains pressed.

#### 6.2.3.1 Handling Incoming Serial Data

One of the criteria for this application note was to make the software less time dependent. By creating a fifo to store incoming characters until the 8051 has time to pro-



cess them, software timing becomes less critical. This application note uses up to 8 levels of the fifo at 9.2KBAUD, and 1 level at 4.8KBAUD and lower. As discussed earlier, the interrupt service routine for the serial port uses the fifo to store incoming data, increments the fifo pointer, "FIFO", and sets SERIALSINT to tell the main program that the fifo needs servicing. Once the main program detects that SERIALSINT is set the routine DECIPHER is executed.

DECIPHER has three separate blocks; a block for decoding displayable characters, a block for processing Escape sequences, and a block for processing Control codes. Each block works on the fifo independently. Before exiting a block, the contents of the fifo are shifted up by the amount of characters that were processed in that particular block. The shifting of the characters insures that the beginning of the fifo contains the next character to be processed. FIFO is then decremented by the number of characters processed.

Let's look at this process more closely. Figure 6.2.1-A shows a representation of a fifo containing 5 characters. The first three characters in the fifo contain displayable characters, A, B, and C respectively with the last two characters being an ESC sequence for moving the cursor up one line (ESC A) and FIFO points to the next available location to be filled by the serial port interrupt routine, in this case, 5.

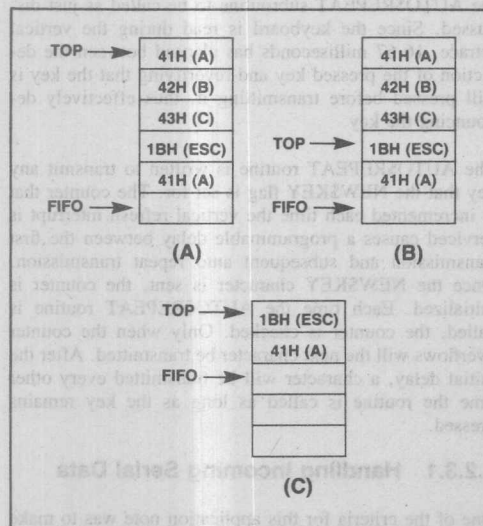


FIGURE 6.2.1 FIFO

When DECIPHER is executed, the first block begins looking at the first character of the fifo for a displayable character. If the character is displayable, it is placed into the display RAM and the software pointer "TOP" that points to the character that is being processed is incremented to the next character. The character is then looked at to see if it too is displayable and if it is, it's placed in the display RAM. The process of checking for displayable characters is continued until either the fifo is empty or a non-displayable character is detected. In our example, three characters are placed into the display RAM before a non-displayable character is detected. At this point the fifo looks like figure 6.2.1-B.

Before entering the next block, the remaining contents of the fifo between TOP, that is now pointing to 1BH and (FIFO-1) are moved up in the fifo by the amount of characters processed, in this example three. TOP is reset to 0 and FIFO is decremented by 3. The serial port interrupt is inhibited during the time the contents of the fifo and the pointers are being manipulated. The fifo now looks like figure 6.2.1-C.

The execution is now passed to the next block that processes ESC sequences. The first location of the fifo is examined to see if it is an ESC character (1BH). If not, the execution is passed to the next block of DECIPHER that processes Control codes. In this case the fifo does contain an ESC code. The flag ESC\$SEQ is checked to see if the 8051 is in the process of receiving an ESC sequence thus signifying that the next byte of the sequence has not been received yet. If the ESC\$SEQ is not set, the next character in the fifo is checked for a valid escape code and the proper subroutine is then called. The fifo contents are then shifted as discussed for the previous block. Due to the length of time that is needed to execute an ESC code sequence or a Control code, only one ESC code and/or Control code can be processed each time DECIPHER is executed.

If at the end of the DECIPHER routine, FIFO contains a 0, the flag SERIALSINT is reset. If SERIALSINT remains set, DECIPHER will be executed immediately after returning to the main program if SCAN had not been set during the execution of the DECIPHER routine, otherwise DECIPHER will be called after the keyboard is read.

## 6.2.4 Memory Pointers and Scrolling

The cursor always points to the next location in display memory to be filled. Each time a character is placed in the display memory, the cursor position needs to be tested to determine if the cursor should be incremented to the beginning of the next line of the display or simply moved to the next position on the current display line. The cursor position pointers are then updated in both the 8276 and the internal registers in the 8051.

When the 2000th character is entered into the display memory, a full display page has been reached signaling the need for the display to scroll. The memory pointer that points to the display memory that contains the first character of the first display line, LINE0, prior to scrolling contains 1800H which is the starting address of the display memory. Each scrolling operation adds 80 (50H) to LINE0 which will now point to the following row in memory as shown in figure 6.2.2-B. LINE0 is used during the vertical

refresh routine to re-initialize the pointers associated with filling the 8276 row buffers.

The display memory locations that were the first line of the CRT display now becomes the last line of the CRT display. Incoming characters are now entered into the display memory starting with 1800H, which is now the first character of the last line of the display screen.

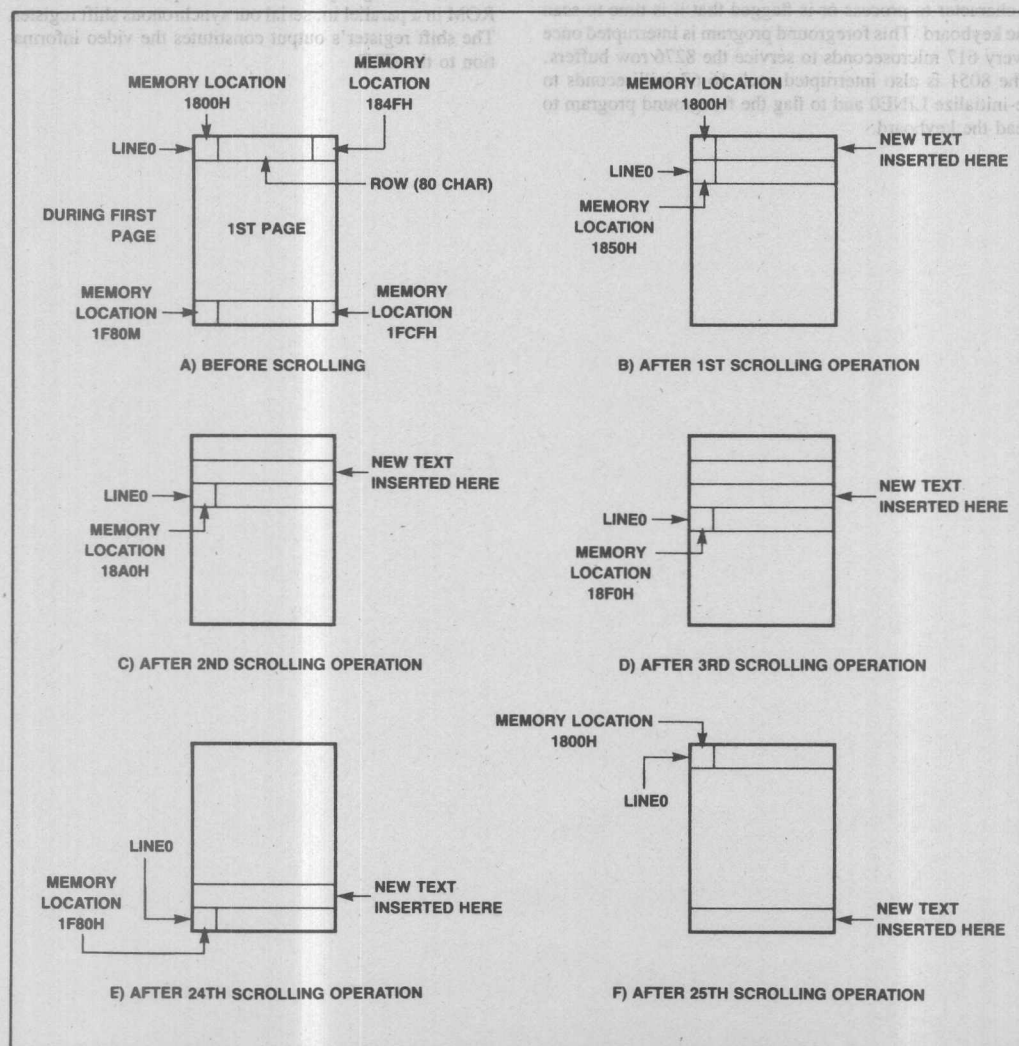


Figure 6.2.2 Pointer Manipulation During Scrolling

### 6.2.5 Software Timing

The use of interrupts to tie the operation of the foreground program to the real-time events of the background program has made the software timing non-critical for this system.

### 6.3 System Operation

Following the system reset, the 8051 initializes all on-chip peripherals along with the 8276 and display ram. After initialization, the processor waits until the fifo has a character to process or is flagged that it is time to scan the keyboard. This foreground program is interrupted once every 617 microseconds to service the 8276 row buffers. The 8051 is also interrupted each 16.67 milliseconds to re-initialize LINE0 and to flag the foreground program to read the keyboard.

As discussed earlier, a special technique of rapidly moving the contents of the display RAM to the 8276 row buffers without the need of a DMA device was employed. The characters are then synchronously transferred to the character generator via CC0-CC6 and LC0-LC2 which are used to display one line at a time. Following the transfer of the first line to the dot timing logic, the line count is incremented and the second line is selected. This process continues until the last line of the character is transferred.

The dot timing logic latches the output of the character ROM in a parallel in, serial out synchronous shift register. The shift register's output constitutes the video information to the CRT.

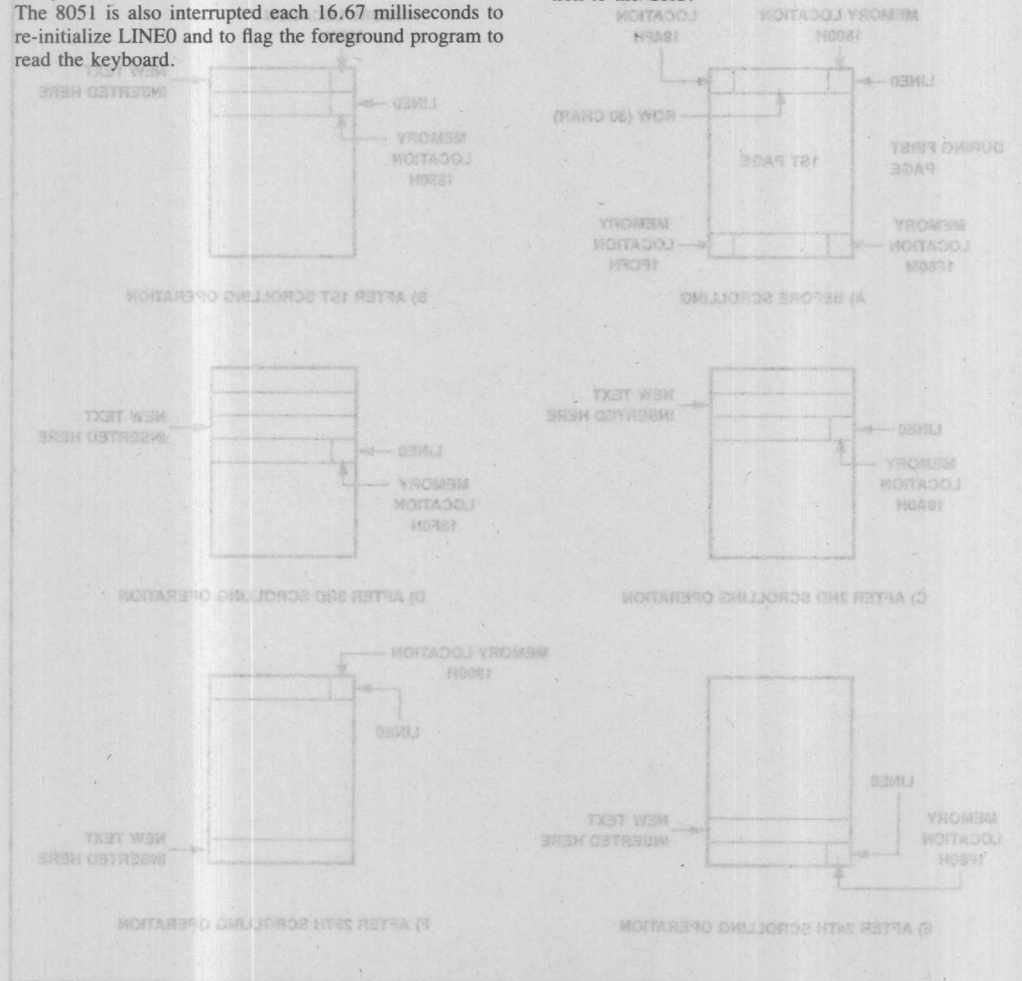
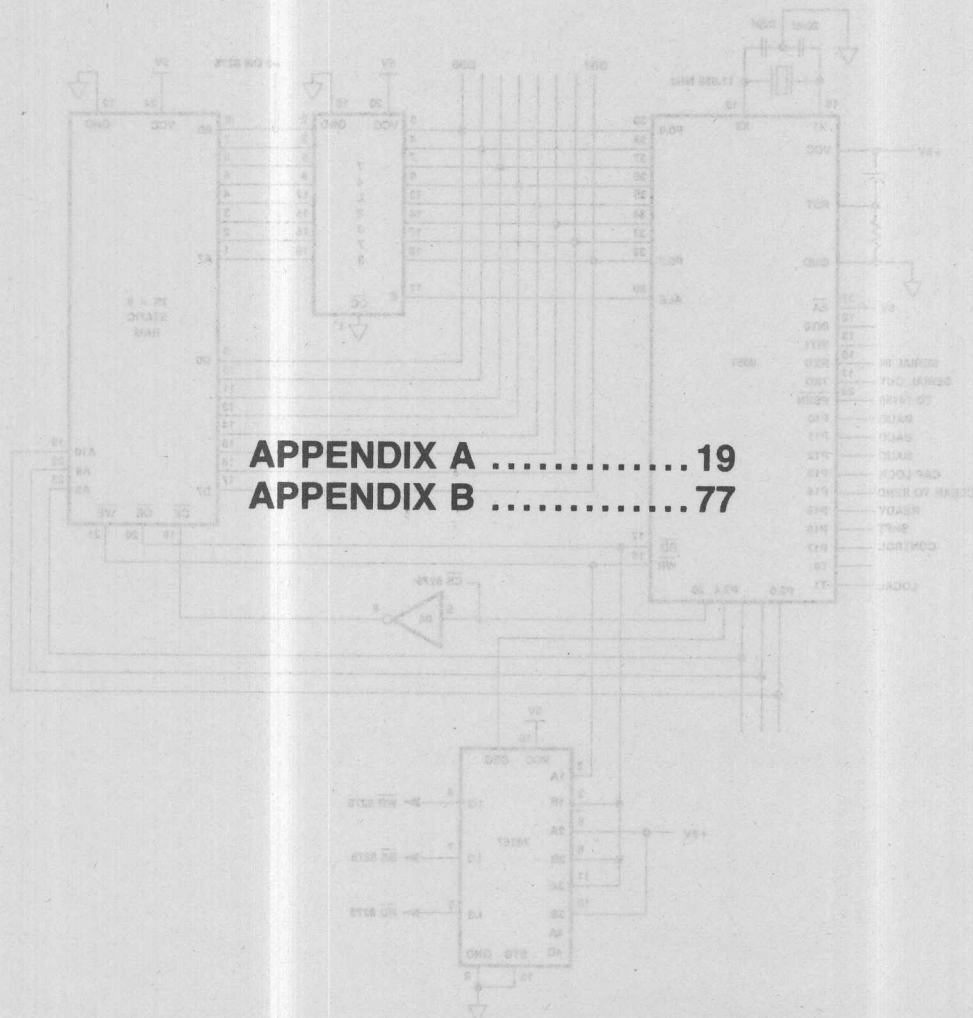
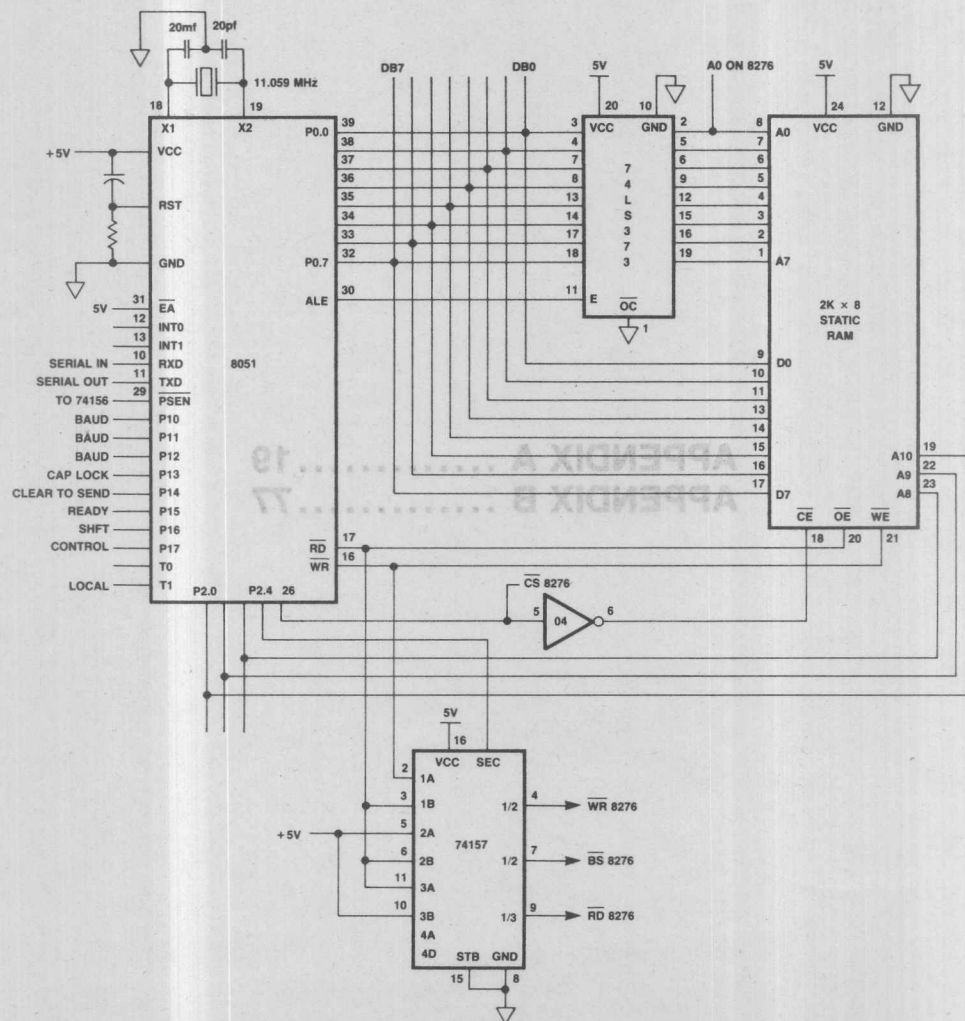


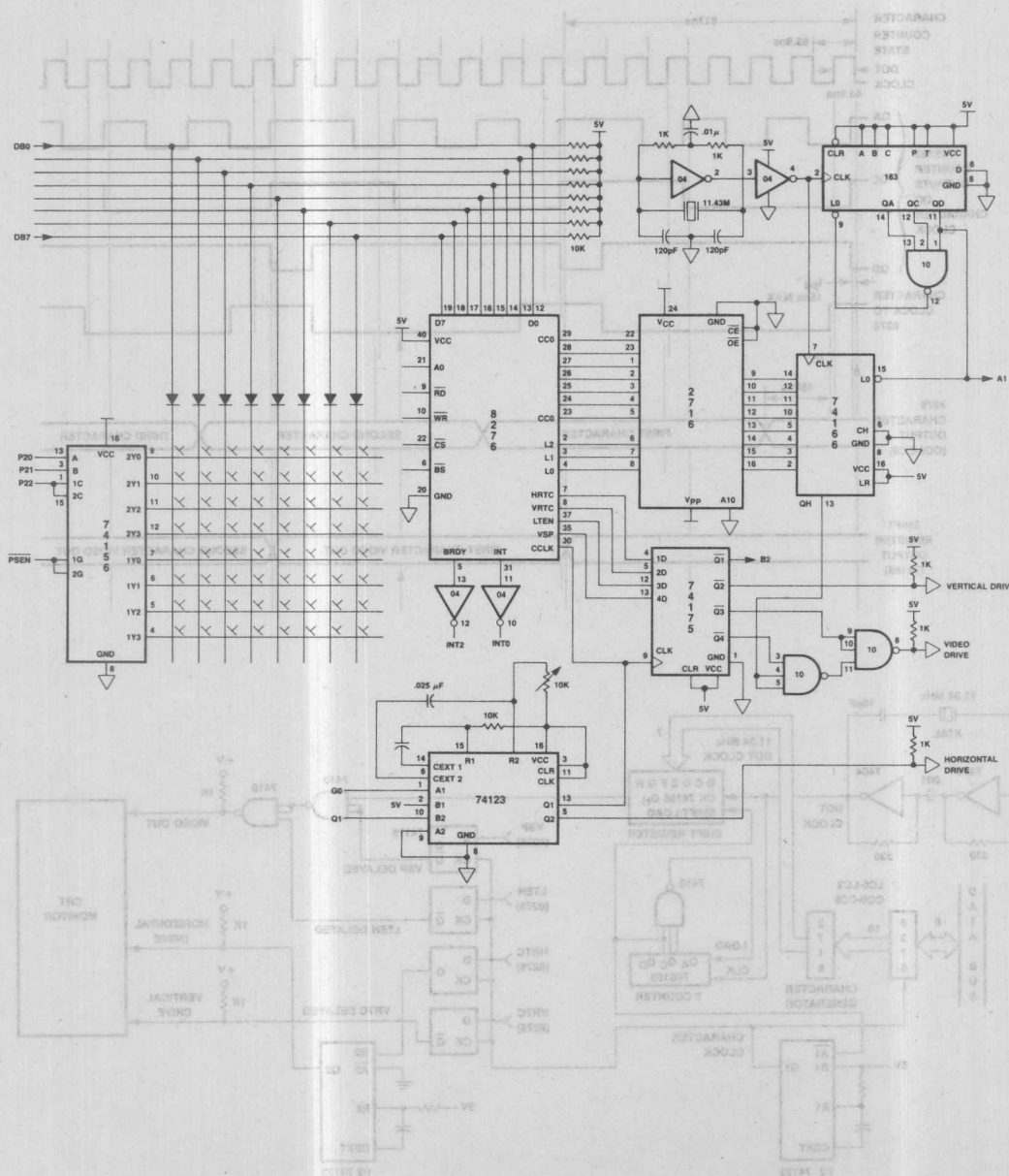
Figure 8.2.2 Pointer Manipulation During Scrolling



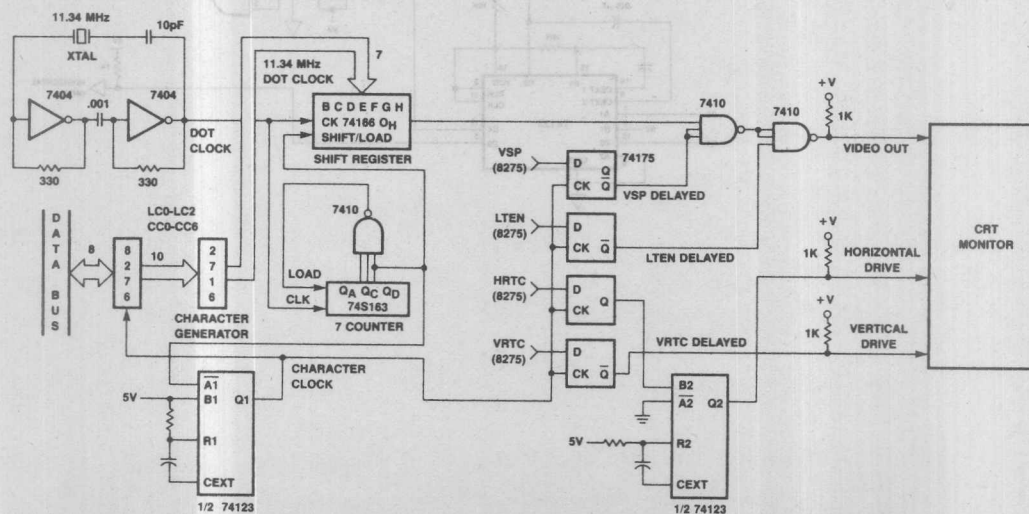
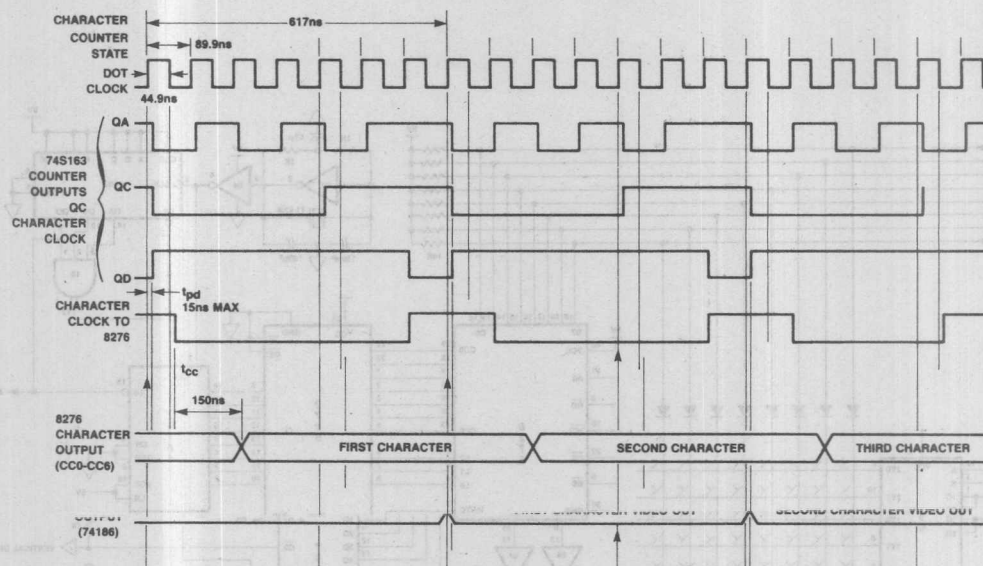


## Appendix 7.1 CRT Schematics

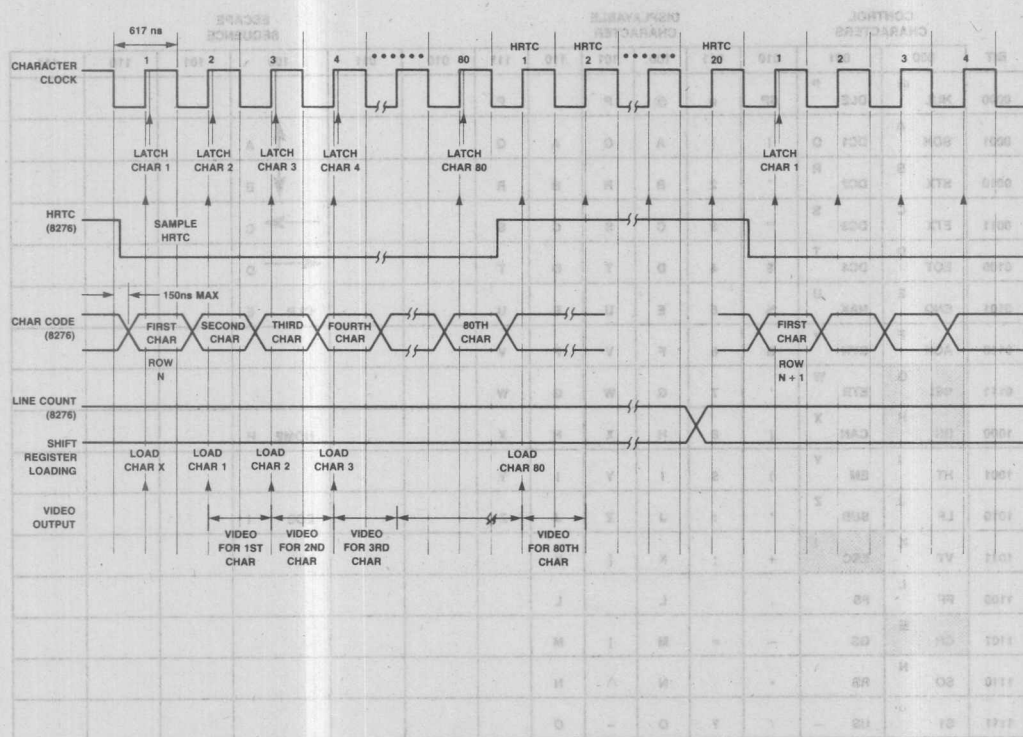




## Appendix 7.2 Dot Timing



# Appendix 7.3 CRT System Timing



NOTE: The timing diagram is a simplified representation of the actual timing. The actual timing is determined by the hardware and software configuration.



## Appendix 7.4 Escape/Control/Display Character Summary

CONTROL CHARACTERS					DISPLAYABLE CHARACTER				ESCAPE SEQUENCE					
BIT	000	001	010	011	100	101	110	111	010	011	100	101	110	111
0000	NUL @	DLE P	SP #	@ P				P						
0001	SOH A	DC1 Q	!		A Q	A Q	A Q				↑ A			
0010	STX B	DC2 R	"	2	B R	B R	B R				↓ B			
0011	ETX C	DC3 S	=	3	C S	C S	C S				→ C			
0100	EOT D	DC4 T	\$	4	D T	D T	D T				← D			
0101	ENQ E	NAK U	%	5	E U	E U	E U				CLR E			
0110	ACK F	SYN V	&	6	F V	F V	F V							
0111	BEL G	ETB W	'	7	G W	G W	G W							
1000	BS H	CAN X	(	8	H X	H X	H X				HOME H			
1001	HT I	EM Y	)	9	I Y	I Y	I Y							
1010	LF J	SUB Z	*	:	J Z	J Z	J Z				EOS I			
1011	VT K	ESC I	+	;	K [	K	K				EL J			
1100	FF L	FS ,			L		L							
1101	CR M	GS -	=		M ]	M	M							
1110	SO N	RS \	.		N ^	N	N							
1111	S1 °	US -	/	?	O -	O	O							

NOTE: Shaded blocks — functions terminal will react to. Others can be generated but are ignored upon receipt.

## Appendix 7.5 Character Generator

As previously mentioned, the character generator used in this terminal is a 2716 EPROM. A 1K by 8 device would have been sufficient since a 128 character 5 by 7 dot matrix only requires 8K of memory. A custom character set could have been stored in the second 1K bytes of the 2716. Any of the free I/O pins on the 8051 could have been used to switch between the character sets.

The three low-order line count outputs (LC0-LC2) from the 8276 are connected to the three low-order address lines of the character generator. The CC0-CC6 output lines are connected to the A3-A9 lines of the character generator.

The output of the character generator is loaded into the shift register. The serial output of the shift register is the video output to the CRT.

Let's assume that the letter "E" is to be displayed. The ASCII code for "E" (45H) is presented to the address lines A2-A9 of the character generator. The scan lines (LC0-LC2) will now count from 0 to seven to form the character as shown in Figure 7.5.0. The same procedure is used to form all 128 possible characters. For reference Appendix 7.6 contains the HEX dump of the character generator used in this terminal.

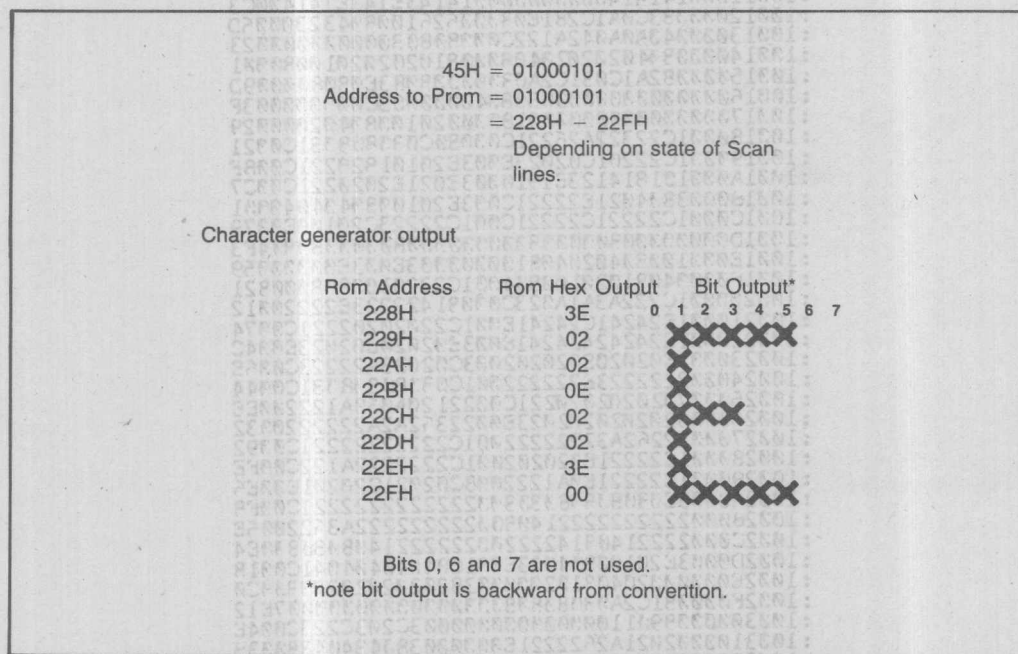


Figure 7.5.0 Character Generator

10-90

## Appendix 7.7 Composite Video

In this design it was assumed that the CRT monitor required a separate horizontal drive, vertical drive, and video input. Many monitors require a composite video signal. The schematic shown in Figure 7.7.0 illustrate how to generate a composite video from the output of the 8276.

The dual one-shots are used to provide a small delay and the proper horizontal and vertical pulse to the composite video monitor. The delay introduced in the horizontal and vertical timing is used to center the display. The 7486 is used to mix the vertical and horizontal retrace. Q1 mix the video and retrace signals along with providing the proper D.C. levels.

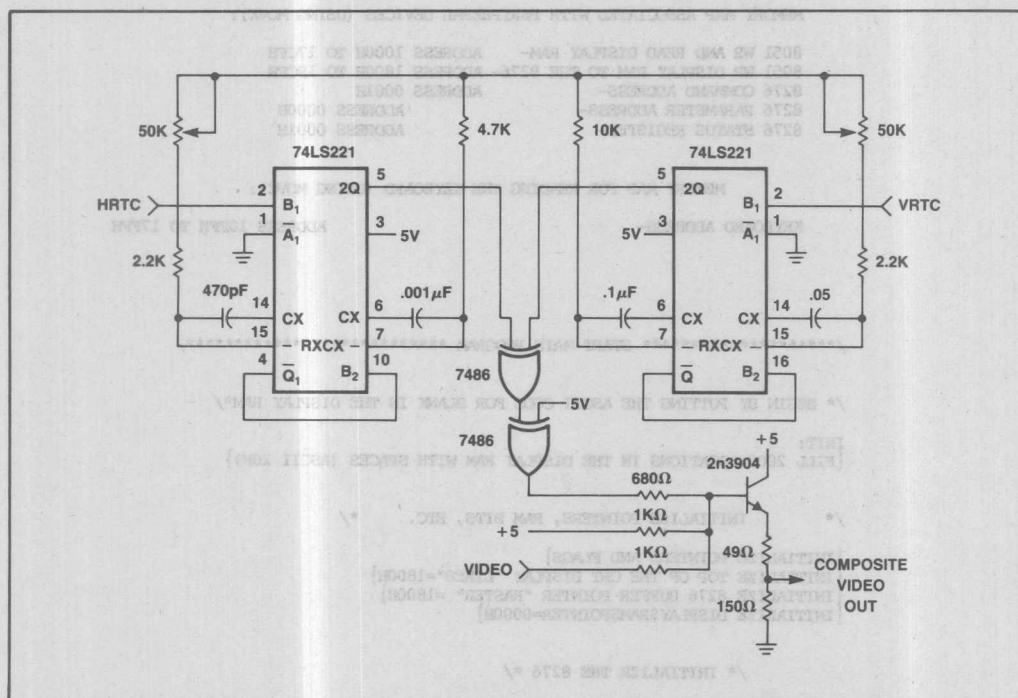


Figure 7.7.0 Composite Video



## Appendix 7.8 Software Documentation

```

/*****
*****
***** SOFTWARE DOCUMENTATION FOR THE 8051 *****
***** TERMINAL CONTROLLER APPLICATION NOTE *****
*****
*****
*****/

```

## MEMORY MAP ASSOCIATED WITH PERIPHERAL DEVICES (USING MOVX):

8051 WR AND READ DISPLAY RAM-	ADDRESS 1000H TO 17CFH
8051 WR DISPLAY RAM TO THE 8276-	ADDRESS 1800H TO 1FCFH
8276 COMMAND ADDRESS-	ADDRESS 0001H
8276 PARAMETER ADDRESS-	ADDRESS 0000H
8276 STATUS REGISTER-	ADDRESS 0001H

## MEMORY MAP FOR READING THE KEYBOARD (USING MOVC):

KEYBOARD ADDRESS-	ADDRESS 10FFH TO 17FFH
-------------------	------------------------

```

/***** START MAIN PROGRAM *****/

```

```

/* BEGIN BY PUTTING THE ASCII CODE FOR BLANK IN THE DISPLAY RAM*/

```

```

INIT:
{FILL 2000 LOCATIONS IN THE DISPLAY RAM WITH SPACES (ASCII 20H)}

```

```

/* INITIALIZE POINTERS, RAM BITS, ETC. */

```

```

{INITIALIZE POINTERS AND FLAGS}
{INITIALIZE TOP OF THE CRT DISPLAY "LINE0"=1800H}
{INITIALIZE 8276 BUFFER POINTER "RASTER" =1800H}
{INITIALIZE DISPLAY$RAM$POINTER=0000H}

```

```

/* INITIALIZE THE 8276 */

```

```

{RESET THE 8276}
{INITIALIZE 8276 TO 80 CHARACTER/ROW }
{INITIALIZE 8276 TO 25 ROWS PER FRAME}
{INITIALIZE 8276 TO 10 LINES PER ROW}
{INITIALIZE 8276 TO NON-BLINKING UNDERLINE CURSER}
{INITIALIZE CURSER TO HOME POSITION (00,00) (UPPER LEFT HAND CORNER)}
{START DISPLAY}
{ENABLE 8276 INTERRUPT}

```

```

/* SET UP 8051 INTERRUPTS AND PRIORITIES */

```

```

{SERIAL PORT HAS HIGHEST INTERRUPT PRIORITY}
{EXTERNAL INTERRUPTS ARE EDGE SENSITIVE}
{ENABLE EXTERNAL INTERRUPTS}

```

/\*PROCEDURE SCANNER: THIS PROCEDURE SCANS THE KEYBOARD AND DETERMINES IF A SINGLE VALID KEY HAS BEEN PUSHED. IF TRUE THEN THE ASCII EQUIVALENT WILL BE TRANSMITTED TO THE HOST COMPUTER.\*/

SCANNER:

{ENABLE 8051 GLOBAL INTERRUPT BIT}

/\* PROGRAMMABLE DELAY FOR THE CURSER BLINK \*/

IF {30 VERTICAL RETRACE INTERRUPTS HAVE OCCURRED (CURSER\$COUNT=1FH)} THEN

DO;

{COMPLEMENT CURSER\$ON}

{CLEAR CURSER\$COUNT}

IF {CURSER IS TO BE OFF (CURSER\$ON=0)} THEN {MOVE CURSER OFF THE SCREEN}

CALL LOAD\$CURSER;

END;

IF {THE LOCAL\$LINE SWITCH HAS CHANGED STATE} THEN

DO;

IF {IN LOCAL MODE} THEN {DISABLE SERIAL PORT INTERRUPT}

ELSE CALL CHECK\$BAUD\$RATE;

END;

DO WHILE {INBETWEEN VERTICAL REFRESHES}

IF {THE FIFO HAS A CHARACTER TO PROCESS (SERIAL\$INT=1)} THEN CALL DECIPHER;

END;

CALL READER;

IF {THE PRESENT PRESSED KEY IS EQUAL TO THE LAST KEY PRESSED AND VALID=1} THEN

CALL AUTO\$REPEAT;

ELSE

DO;

IF {A KEY IS PRESSED BUT NOT THE SAME ONE AS THE LAST KEYBOARD SCAN} THEN

DO;

IF {ONLY ONE KEY IS PRESSED} THEN

{GET THE ASCII CODE FOR IT}

{SET NEWSKEY AND VALID FLAGS}

ELSE {RESET VALID AND NEWSKEY FLAGS}

END;

ELSE {THE KEYBOARD MUST NOT HAVE A KEY PRESSED SO RESET VALID\$KEY AND NEWSKEY FLAGS}

END;

GOTO SCANNER;

END;

/\* PROCEDURE AUTO\$REPEAT: THIS PROCEDURE WILL PERFORM AN AUTO REPEAT FUNCTION BY TRANSMITTING A CHARACTER EVERY OTHER TIME THIS ROUTINE IS CALLED. THE AUTO REPEAT FUNCTION IS ACTIVATED AFTER A FIXED DELAY PERIOD AFTER THE FIRST CHARACTER IS SENT\*/

AUTO\$REPEAT:

IF {THE KEY PRESSED IS NEW (NEWSKEY=1)} THEN

DO;

{CLEAR THE DIVIDE BY TWO COUNTER "TRANSMIT\$TOGGLE"}

{INITIALIZE THE DELAY COUNTER "TRANSMIT\$COUNT" TO 000H}

CALL TRANSMIT;

{CLEAR NEWSKEY}

/\* FIRST CHARACTER \*/

END;

```

ELSE
DO;
  IF {TRANSMIT$COUNT IS NOT EQUAL TO 0} THEN
  DO;
    {INCREMENT TRANSMIT$COUNT}
    IF TRANSMIT$COUNT=0FFH THEN
    DO;
      CALL TRANSMIT;
      {CLEAR TRANSMIT$COUNT}
    END;
  END;
ELSE
DO;
  {TURN THE CURSER ON DURING THE AUTO REPEAT FUNCTION}
  IF TRANSMIT$TOGGLE = 1 THEN
  DO;
    CALL TRANSMIT;
    {COMPLEMENT TRANSMIT$TOGGLE}
  END;
END;
END AUTO$REPEAT;

```

/\* PROCEDURE TRANSMIT- ONCE THE HOST COMPUTER SIGNALS THE 8051H BY BRINGING THE CLEAR-TO-SEND LINE LOW, THE ASCII CHARACTER IS PUT INTO THE SERIAL PORT.\*/

```

TRANSMIT:
PROCEDURE;
IF {THE TERMINAL IS ON-LINE} THEN
DO;
  {WAIT UNTIL THE CLEAR$TO$SEND LINE IS LOW AND UNTIL THE 8051 SERIAL PORT TX IS NOT BUSY (TRANSMIT$INT=1)}
  {TRANSMIT THE ASCII CODE}
  {CLEAR THE FLAG "TRANSMIT$INT". THE SERIAL PORT SERVICE ROUTINE WILL SET THE FLAG WHEN THE SERIAL PORT IS FINISHED TRANSMITTING}
END;
ELSE {THE TERMINAL IS IN THE LOCAL MODE}
DO;
  {PUT THE ASCII CODE IN THE FIFO}
  {INCREMENT THE FIFO POINTER}
  {SET SERIAL$INT}
END;
END TRANSMIT;

```

/\* PROCEDURE DECIPHER: THIS PROCEDURE DECODES THE HOST COMPUTER'S MESSAGES AND DETERMINES WHETHER IT IS A DISPLAYABLE CHARACTER, CONTROL SEQUENCE, OR AN ESCAPE SEQUENCE THE PROCEDURE THEN ACTS ACCORDINGLY \*/

DECIPHER:  
START\$DECIPHER:

VALID\$RECEPTION=0;  
DO WHILE {THE FIFO IS NOT EMPTY AND THE CHARACTER IS DISPLAYABLE}  
  RECEIVE={ASCII CODE}  
  CALL DISPLAY;  
  {NEXT CHARACTER}  
END;

IF {CHARACTERS WERE DISPLAYED} THEN  
  {DISABLE SERIAL PORT INTERRUPT}  
  MOVE THE REMAINING CONTENTS OF THE FIFO UP TO THE BEGINNING OF THE FIFO  
  {ENABLE SERIAL PORT INTERRUPT}  
  SET THE "VALID\$RECEPTION" FLAG

IF {THE FIFO IS EMPTY} THEN {CLEAR THE "SERIAL\$INT FLAG AND RETURN}

IF {THE NEXT CHARACTER IS AN "ESC" CODE } THEN

DO:  
  {LOOK AT THE CHARACTER IN THE FIFO AFTER THE ESC CODE AND CALL THE CORRECT SUBROUTINE}

  ;  
  CALL UP\$CURSER;                               /\* ESC A \*/  
  CALL DOWN\$CURSER;                            /\* ESC B \*/  
  CALL RIGHT\$CURSER;                           /\* ESC C \*/  
  CALL LEFT\$CURSER;                            /\* ESC D \*/  
  CALL CLEAR\$SCREEN;                           /\* ESC E \*/  
  CALL MOV\$CURSER;                             /\* ESC F \*/

  ;  
  CALL HOME;                                   /\* ESC H \*/  
  ;  
  CALL ERASE\$FROM\$CURSER\$TO\$END\$OF\$SCREEN;   /\* ESC J \*/  
  CALL BLINK;                                  /\* ESC K \*/

  {DISABLE THE SERIAL PORT INTERRUPT}  
  MOVE THE REMAINING CONTENTS OF THE FIFO UP TO THE BEGINNING OF THE FIFO  
  {ENABLE THE SERIAL PORT INTERRUPT}  
  SET THE "VALID\$RECEPTION" FLAG

IF {THE FIFO IS EMPTY} THEN {CLEAR THE SERIAL\$INT FLAG AND RETURN}  
END;



```

IF {THE NEXT CHARACTER IS A CONTROL CODE} THEN
DO;
  CALL THE RIGHT SUBROUTINE;

  CALL LEFT$CURSER; /* CTL H */
  CALL LINE$FEED; /* CTL J */
  CALL CLEAR$SCREEN; /* CTL L */
  CALL CARRIAGE$RETURN; /* CTL M */

  {DISABLE THE SERIAL PORT INTERRUPT}
  {MOVE THE REMAINING CONTENTS OF THE FIFO UP TO THE BEGINNING OF THE FIFO}
  {ENABLE THE SERIAL PORT INTERRUPT}
  {SET THE "VALID$RECEPTION" FLAG}
END;

IF {NO VALID CODE WAS RECEIVED ("VALID$RECEPTION" IS 0)} THEN
  {THROW THE CHARACTER OUT AND MOVE THE REMAINING CONTENTS OF THE FIFO}
  {UP TO THE BEGINNING}

IF {THE FIFO IS EMPTY} THEN {CLEAR THE SERIAL$INT FLAG AND RETURN}

END DECIPHER;

/* PROCEDURE DISPLAY: THIS PROCEDURE WILL TAKE THE BYTE IN RAM LABELED
RECEIVE AND PUT IT INTO THE DISPLAY RAM. */

DISPLAY:
  {PUT INTO THE DISPLAY RAM LOCATION POINTED TO BY "DISPLAY$RAM$POINTER"
  THE CONTENTS OF RECEIVE}

  IF {THE END OF THE DISPLAY MEMORY HAS BEEN REACHED} THEN
    {RESET "DISPLAY$RAM$POINTER" TO THE BEGINNING OF THE RAM}
  ELSE
    {INCREMENT "DISPLAY$RAM$POINTER"}

  IF {THE CURSER IS IN THE LAST COLUMN OF THE CRT DISPLAY} THEN
  DO;
    {MOVE THE CURSER BACK TO THE BEGINNING OF THE LINE}
    IF {THE NEW DISPLAY RAM LOCATION HAS A END-OF-LINE CHARACTER IN IT} THEN
      CALL FILL;

    IF {THE CURSER IS ON THE LAST LINE OF THE CRT DISPLAY} THEN
      CALL SCROLL;
    ELSE
      {MOVE THE CURSER TO THE NEXT LINE}
  END;
  ELSE
    {INCREMENT THE CURSER TO THE NEXT LOCATION}

  {TURN THE CURSER ON }
  CALL LOADCURSER;
END DISPLAY;

```

```

/*      PROCEDURE LINE$FEED      */
PROCEDURE HOME: THIS PROCEDURE MOVES THE CURSOR TO THE 0,0
LINE$FEED:
IF {THE CURSOR IS IN THE LAST LINE OF THE CRT DISPLAY} THEN
    CALL SCROLL;
ELSE
    DO;
        {MOVE THE CURSOR TO THE NEXT LINE}
        {TURN THE CURSOR ON}
        CALL LOAD$CURSER;
    END;
IF {THE DISPLAY$RAM$POINTER IS ON THE LAST LINE IN THE DISPLAY RAM} THEN
    {MOVE THE DISPLAY$RAM$POINTER TO THE FIRST LINE IN THE DISPLAY RAM}
ELSE
    {MOVE THE DISPLAY$RAM$POINTER TO THE NEXT LINE IN THE DISPLAY RAM}
IF {THE FIRST CHARACTER IN THE NEW LINE CONTAINS AN END-OF-LINE CHARACTER } THEN
    CALL FILL;
END LINE$FEED;

```

```

/*      PROCEDURE SCROLL      */
SCROLL:
CALL BLANK;
{DISABLE VERTICAL RETRACE INTERRUPT}
IF {THE FIRST LINE OF THE CRT CONTAINS THE LAST LINE OF THE DISPLAY MEMORY} THEN
    {MOVE THE POINTER "LINE0" TO THE BEGINNING OF THE DISPLAY MEMORY}
ELSE
    {MOVE "LINE0" TO THE NEXT LINE IN THE DISPLAY MEMORY}
{ENABLE VERTICAL RETRACE INTERRUPT}
END SCROLL;

```

```

/*      PROCEDURE CLEAR SCREEN      */
CLEAR$SCREEN:
CALL HOME;
CALL ERASE$FROM$CURSER$TO$END$OF$SCREEN;
END CLEAR$SCREEN;

```

/\* PROCEDURE HOME: THIS PROCEDURE MOVES THE CURSER TO THE 0,0 POSITION \*/

HOME:

{MOVE THE CURSER POSITION TO THE UPPER LEFT HAND CORNER OF THE CRT}  
 {TURN THE CURSER ON}  
 CALL LOAD\$CURSER;  
 {MOVE THE DISPLAY\$RAM\$POINTER TO THE CORRECT LOCATION IN THE DISPLAY RAM}

END HOME;

/\* PROCEDURE ERASE FROM CURSER TO END OF SCREEN: \*/

ERASE\$FROM\$CURSER\$TO\$END\$OF\$SCREEN:

CALL BLINE;

/\* ERASE CURRENT LINE \*/

IF {THE CURSER IS NOT ON THE LAST LINE OF THE CRT DISPLAY} THEN  
 STARTING WITH THE NEXT LINE, PUT AN END-OF-LINE CHARACTER (0F1H)  
 IN THE DISPLAY RAM LOCATIONS THAT CORRESPOND TO THE BEGINNING OF  
 THE CRT DISPLAY LINES UNTIL THE BOTTOM OF THE CRT SCREEN HAS BEEN REACHED)

END;

END ERASE\$FROM\$CURSER\$TO\$END\$OF\$SCREEN;

/\*PROCEDURE MOV\$CURSER: THIS PROCEDURE IS USED IN CONJUNCTION WITH WORDSTAR  
 IF A ESC F IS RECEIVED FROM THE HOST COMPUTER, THE TERMINAL CONTROLLER WILL  
 READ THE NEXT TWO BYTE TO DETERMINE WHERE TO MOVE THE CURSER. THE FIRST BYTE  
 IS THE ROW INFORMATION FOLLOWED BY THE COLUMN INFORMATION \*/

MOV\$CURSER:

{WAIT UNTIL THE FIFO HAS RECEIVED THE NEXT TWO CHARACTERS}  
 {MOVE THE CURSER TO THE LOCATION SPECIFIED IN THE ESCAPE SEQUENCE}  
 {MOVE THE DISPLAY\$RAM\$POINTER TO THE CORRECT LOCATION}

IF THE FIRST CHARACTER IN THE NEW LINE HAS AN END-OF-LINE CHARACTER} THEN  
 CALL FILL;  
 END;

{DISABLE THE SERIAL PORT INTERRUPT}  
 {MOVE THE REMAIN CONTENTS OF THE FIFO UP TWO LOCATIONS IN MEMORY}  
 {DECREMENT THE FIFO BY TWO}  
 {ENABLE THE SERIAL PORT INTERRUPT}

END MOV\$CURSER;

/\* PROCEDURE LEFT CURSER: THIS PROCEDURE MOVES THE CURSER LEFT ONE COLUMN  
 BY SUBTRACTING 1 OF THE CURSER COLUMN RAM LOCATION THEN CALL LOAD CURSER \*/

LEFT\$CURSER:

IF {THE CURSER IS NOT IN THE FIRST LOCATION OF A LINE} THEN  
 DO;  
 {MOVE THE CURSER LEFT BY ONE LOCATION}  
 {TURN THE CURSER ON}  
 CALL LOAD\$CURSER;  
 {DECREMENT THE DISPLAY\$RAM\$POINTER BY ONE}

END;

END LEFT\$CURSER;

```

/*      PROCEDURE RIGHT CURSER: THIS PROCEDURE MOVES THE CURSER RIGHT ONE COLUMN
      BY ADDING 1 TO THE CURSER COLUMN RAM LOCATION THEN CALL LOAD CURSER */

```

```

RIGHT$CURSER:

```

```

IF {THE CURSER IS NOT IN THE LAST POSITION OF THE CRT LINE} THEN
DO;
  {MOVE THE CURSER RIGHT BY ONE LOCATION}
  {TURN THE CURSER ON}
  CALL LOAD$CURSER;
  {INCREMENT THE DISPLAY$RAM$POINTER BY ONE}
END;

END RIGHT$CURSER;

```

```

/*      PROCEDURE UP CURSER: THIS PROCEDURE MOVES THE CURSER UP ONE ROW
      BY SUBTRACTING 1 TO THE CURSER ROW RAM LOCATION THEN CALL LOAD CURSER */

```

```

UP$CURSER:

```

```

IF {THE CURSER IS NOT ON THE FIRST LINE OF THE CRT DISPLAY} THEN
DO;
  {MOVE THE CURSER UP ONE LINE}
  {TURN ON THE CURSER}
  CALL LOAD$CURSER;

  IF {THE DISPLAY$RAM$POINTER IS IN THE FIRST LINE OF DISPLAY MEMORY} THEN
  {MOVE THE DISPLAY$RAM$POINTER TO THE LAST LINE OF DISPLAY MEMORY}
  ELSE
  {MOVE THE DISPLAY$RAM$POINTER UP ONE LINE IN DISPLAY MEMORY}

  IF {THE FIRST LOCATION OF THE NEW LINE CONTAINS AN END-OF-LINE CHARACTER} THEN
  CALL FILL;
END;

END UP$CURSER;

```

```

/*      PROCEDURE DOWN CURSER: THIS PROCEDURE MOVES THE CURSER DOWN ONE ROW
      BY ADDING 1 TO THE CURSER ROW RAM LOCATION THEN CALL LOAD CURSER */

```

```

DOWN$CURSER:

```

```

IF {THE CURSER IS NOT ON THE LAST LINE OF THE CRT DISPLAY} THEN
DO;
  {TURN THE CURSER ON}
  {MOVE THE CURSER TO THE NEXT LINE}
  CALL LOAD$CURSER;

  IF {THE DISPLAY$RAM$POINTER IS NOT ON THE LAST LINE OF THE DISPLAY MEMORY} THEN
  {MOVE THE DISPLAY$RAM$POINTER TO THE NEXT LINE IN THE DISPLAY MEMORY}
  ELSE
  {MOVE THE DISPLAY$RAM$POINTER TO THE FIRST LINE IN THE DISPLAY MEMORY}

  IF {THE FIRST CHARACTER IN THE NEW LINE IS AN END-OF-LINE CHARACTER} THEN
  CALL FILL;
END;

END DOWN$CURSER;

```



```

/*      PROCEDURE CARRIAGE$RETURN
CARRIAGE$RETURN:
{MOVE THE DISPLAY$RAM$POINTER TO THE BEGINNING OF THE CURRENT LINE IN THE DISPLAY MEMORY}
{MOVE THE CURSER TO THE BEGINNING OF THE CURRENT LINE OF THE CRT DISPLAY}
{TURN THE CURSER ON}
CALL LOAD$CURSER;

END CARRIAGE$RETURN;

/*      PROCEDURE LOAD CURSER: LOAD CURSER TAKES THE VALUE HELD IN RAM AND
LOADS IT INTO THE 8276 CURSER REGISTER. */

LOAD$CURSER:
PROCEDURE;
IF {THE CURSER IS ON} THEN
{MOVE THE CURSER BACK ONTO THE CRT DISPLAY}
{DISABLE BUFFER INTERRUPT}
{WRITE TO THE 8276 CURSER REGISTERS THE X,Y LOCATIONS}
{ENABLE BUFFER INTERRUPT}

END LOAD$CURSER;

/*      PROCEDURE CHECK BAUD RATE: THIS PROCEDURE READS THE THREE PORT PINS ON P1 AND SETS UP
THE SERIAL PORT FOR THE SPECIFIED BAUD RATE */

CHECK$BAUD$RATE:
{SET TIMER 1 TO MODE 1 AND AUTO RELOAD}
{TURN TIMER ON}
{ENABLE SERIAL PORT INTERRUPT}
{READ BAUD RATE SWITCHES AND SET UP RELOAD VALUE}

;
TH1=040H;
TH1=0A0H;
TH1=0D0H;
TH1=0E8H;
TH1=0F4H;
TH1=0FAH;
TH1=0FDH;

/* 00 IS NOT ALLOWED */
/* 150 BAUD */
/* 300 BAUD */
/* 600 BAUD */
/* 1200 BAUD */
/* 2400 BAUD */
/* 4800 BAUD */
/* 9600 BAUD */

END CHECK$BAUD$RATE;

```

```

/*      PROCEDURE READER: THIS PROCEDURE IS WRITTEN IN ASSEMBLY LANGUAGE. THE
EXTERNAL PROCEDURE SCANS THE 8 LINES OF THE KEYBOARD AND READS THE RETURN
LINES. THE STATUS OF THE 8 RETURN LINES ARE THEN STORED IN INTERNAL
MEMORY ARRAY CALLED CURRENT$KEY */

```

READER:

```

{INITIALIZE FLAGS "KEY0"=0, "SAME"=1, 0 COUNTER=0}

```

```

DO UNTIL {ALL 8 KEYBOARD SCAN LINES ARE READ}

```

```

  {READ KEYBOARD SCAN}

```

```

  IF {NO KEY WAS PRESSED} THEN

```

```

    {INCREMENT 0 COUNTER}

```

```

  ELSE

```

```

    IF {THE KEY PRESSED WAS NOT THE SAME KEY THAT WAS PRESSED THE LAST TIME
        THE KEYBOARD WAS READ} THEN

```

```

      {CLEAR "SAME" AND WRITE NEW SCAN RESULT TO CURRENT$KEY RAM ARRAY}

```

```

  END;

```

```

  IF {ALL 8 SCANS DIDN'T HAVE A KEY PRESSED (0 COUNTER=8)} THEN
    {SET KEY0, AND CLEAR SAME}

```

```

END READER;

```

```

/*      PROCEDURE BLANK: THIS EXTERNAL PROCEDURE FILLS LINE0 WITH SPACES (20H ASCII)
DURING THE SCROLL ROUTINES.*/

```

BLANK:

```

DO I= {BEGINNING OF THE CRT DISPLAY (LINE0)} TO {LINE0 + 50H}

```

```

  {DISPLAY RAM POINTED TO BY "I" = SPACE (ASCII 20H)}

```

```

  NEXT I

```

```

END;

```

```

END BLANK;

```

```

/*      PROCEDURE BLINE: THIS EXTERNAL PROCEDURE BLANKS FROM THE CURSER TO THE END OF
THE DISPLAY LINE */

```

BLINE:

```

DO I= {CURRENT CURSER POSITION ON CRT DISPLAY} TO {END OF ROW}

```

```

  {DISPLAY RAM POINTED TO BY "I" = SPACE (ASCII 20H)}

```

```

  NEXT I

```

```

END;

```

```

END BLINE;

```

```

/*      PROCEDURE FILL: THIS EXTERNAL PROCEDURE FILLS A DISPLAY LINE WITH SPACES*/

```

FILL:

```

DO I= {BEGINNING OF THE LINE THAT THE CURSER IS ON} TO {END OF THE ROW}

```

```

  {DISPLAY RAM POINTED TO BY "I" = SPACE (ASCII 20H)}

```

```

  NEXT I

```

```

END;

```

```

END FILL;

```

CFH - 100-100000-100000

WOLFGANG ELLNER: THIS DATE

TASM.OBJ,DECODE.OBJ,PLM51.LIB

PL/M-51 COMPILER

PL/M-51 COMPILER

```

SELECT
CRT$CONTROLLER:
DO;

/***** REGISTER TIE *****/
/***** BYTE REG *****/
/***** DECLARE LITERALS *****/
DECLARE LLC LITERALLY 'LOCAL$LINE$CHANGE';
DECLARE REG LITERALLY 'REGISTER';
DECLARE CURRENT$KEY LITERALLY 'CURKEY';
DECLARE SERIAL$SERVICE LITERALLY 'SERBUF';
DECLARE DISPLAY$RAM$POINTER LITERALLY 'POINT';
DECLARE SERIAL$INT LITERALLY 'SERINT';
DECLARE TRANSMIT$INT LITERALLY 'TRNINT';
DECLARE CURSER$COLUMN LITERALLY 'CURSER';
DECLARE LAST$KEY LITERALLY 'LSIKEY';
DECLARE CURSER$COUNT LITERALLY 'COUNT';
DECLARE SCAN$INT LITERALLY 'SCAN';

/***** REGISTER DECLARATIONS FOR THE 8051 *****/
/***** BYTE REGISTERS *****/
DECLARE
P0 BYTE AT (80H) REG,
P1 BYTE AT (90H) REG,
P2 BYTE AT (0A0H) REG,
P3 BYTE AT (0B0H) REG,
PSW BYTE AT (0D0H) REG,
ACC BYTE AT (0E0H) REG,
B BYTE AT (0F0H) REG,
SP BYTE AT (81H) REG,
DPL BYTE AT (82H) REG,
DPH BYTE AT (83H) REG,
PCON BYTE AT (87H) REG,
TCON BYTE AT (88H) REG,
TMOD BYTE AT (89H) REG,
TLO BYTE AT (8AH) REG,
TL1 BYTE AT (8BH) REG,
TH0 BYTE AT (8CH) REG,
TH1 BYTE AT (8DH) REG,
IE BYTE AT (0A8H) REG,
IP BYTE AT (0B8H) REG,
SCON BYTE AT (98H) REG,
SBUF BYTE AT (99H) REG;

/***** REGISTER TIE *****/
/***** BYTE REG *****/
/***** DECLARE LITERALS *****/
/***** REGISTER DECLARATIONS FOR THE 8051 *****/
/***** BYTE REGISTERS *****/

```



PL/M-51 COMPILER CRTCONTROLLER

PL/M-51 COMPILER

```

$EJECT
/***** BIT REGISTERS *****/

/***** PSW BITS *****/
14 1 DECLARE
CY BIT AT(0D7H) REG,
AC BIT AT(0D6H) REG,
FO BIT AT(0D5H) REG,
RS1 BIT AT(0D4H) REG,
RS0 BIT AT(0D3H) REG,
OV BIT AT(0D2H) REG,
P BIT AT(0D0H) REG,

/***** TOON BITS *****/
TF1 BIT AT(8FH) REG,
TR1 BIT AT(8EH) REG,
TF0 BIT AT(8DH) REG,
TR0 BIT AT(8CH) REG,
IE1 BIT AT(8BH) REG,
IT1 BIT AT(8AH) REG,
IE0 BIT AT(89H) REG,
IT0 BIT AT(88H) REG,

/***** IE BITS *****/
EA BIT AT(0AFH) REG,
ES BIT AT(0ACH) REG,
ET1 BIT AT(0ABH) REG,
EX1 BIT AT(0AAH) REG,
ET0 BIT AT(0A9H) REG,
EX0 BIT AT(0A8H) REG,

/***** IP BITS *****/
PS BIT AT(0BCH) REG,
PT1 BIT AT(0BBH) REG,
PX1 BIT AT(0BAH) REG,
PT0 BIT AT(0B9H) REG,
PX0 BIT AT(0B8H) REG,

/***** P3 BITS *****/
RD BIT AT(0B7H) REG,
WR BIT AT(0B6H) REG,
T1 BIT AT(0B5H) REG,
T0 BIT AT(0B4H) REG,
INT1 BIT AT(0B3H) REG,
INT0 BIT AT(0B2H) REG,
TKD BIT AT(0B1H) REG,
RKD BIT AT(0B0H) REG,

/***** SOON BITS *****/
SM0 BIT AT(9FH) REG,
SM1 BIT AT(9EH) REG,
SM2 BIT AT(9DH) REG,
REN BIT AT(9CH) REG,
TB8 BIT AT(9BH) REG,
RB8 BIT AT(9AH) REG,
TI BIT AT(99H) REG,
RI BIT AT(98H) REG,

```

PL/M-51 COMPILER CRTCONTROLLER

```

$EJECT
$IF SW1
/***** DECLARE CONSTANTS *****/

15 1 DECLARE LOW$SCAN(16) STRUCTURE
      (KEY (8) BYTE) CONSTANT

      ('890-',5CH,5EH,08H,00H,
/* SCAN 0, SHIFT KEY =0; 8,9,0,-,\,^, BACK SPACE */
      'uiop',5BH,'@',0AH,7FH,
/* SCAN 1, SHIFT =0; u,i,o,p[,e, LINE FEED, DELETE */
      'jkl;:',00H,0DH,'7',
/* SCAN 2, SHIFT =0; j,k,l,;,:, RETURN, 7 */
      'm',2CH,'.',00H,'/',00H,00H,00H,
/* SCAN 3, SHIFT =0; m,COMMA,.,/ */
      00H,'azxcvbn',
/* SCAN 4, SHIFT =0; a,z,x,c,v,b,n */
      'y',00H,00H,'dfgh',
/* SCAN 5, SHIFT =0; y, SPACE, d,f,g,h */
      09H,'qwsert',00H,
/* SCAN 6, SHIFT =0; TAB,q,w,s,e,r,t */
      1BH,'123456',00H,
/* SCAN 7, SHIFT =0;ESC,1,2,3,4,5,6 */
      28H,29H,00H,'=',7CH,7EH,08H,00H,
/* SCAN 0, SHIFT =1; (,),=,|,~, BACK SPACE */
      'UIOP',00H,00H,0AH,7FH,
/* SCAN 1, SHIFT =1; U,I,O,P, LINE FEED, DELETE */
      'JKL+*',00H,0DH,27H,
/* SCAN 2, SHIFT =1; J,K,L,+,*, RETURN, ' */
      'M<>',00H,3FH,00H,00H,00H,
/* SCAN 3, SHIFT =1; M,<,>,* */
      00H,'AZXCVBN',
/* SCAN 4, SHIFT =1; A,Z,X,C,V,B,N */
      'Y',00H,00H,'DFGH',
/* SCAN 5, SHIFT =1; Y, SPACE, D,F,G,H */
      09H,'QWERT',00H,
/* SCAN 6, SHIFT =1; TAB, Q,W,S,E,R,T */
      1BH,'!"$%&',00H);
/* SCAN 7, SHIFT =1;ESC,!,",$,%,& */
$ENDIF

```

CRTCNTROLLER

```

16 1 DECLARE
    $IF SW2
        INPUT BIT AT (0B4H) REG,
        $ENDIF
        $IF SW1
            CAP$LOCK BIT AT (095H) REG,
            SHIFT$KEY BIT AT (096H) REG,
            CONTROL$KEY BIT AT (097H) REG,
            $ENDIF
            LOCAL$LINE BIT AT (0B5H) REG,
            CLEAR$TO$SEND BIT AT (093H) REG,
            DATA$TERMINAL$READY BIT AT (094H) REG,
17 1 DECLARE (
    $IF SW1
        SAME,
        VALID$KEY,
        KEY0,
        LAST$SHIFT$KEY,
        LAST$CONTROL$KEY,
        LAST$CAP$LOCK,
    $ENDIF
    $IF SW2
        RCV$FLG,
        SYNC,
        BY$FIN,
        KBD$INT,
        ERROR,
    $ENDIF
        NEWS$KEY,
        TRANSMIT$TOGGLE,
        CURSER$ON,
        SERIAL$INT,
        SCAN$INT,
        TRANSMIT$INT,
        ESC$SEQ,
        VALID$RECEPTION,
        LLC,
        ENSP) BIT PUBLIC;

```

		SYMBOL	ADDRESS	VALUE
18	1	DECLARE (		
		ASCII\$KEY,		
		TRANSMIT\$COUNT,		
		TEMP,		
		SHIFT,		
		CURSER\$COL,		
		CURSER\$COLUMN,		
		CURSER\$ROW,		
		CURSER\$COUNT,		
		FIFO,		
		RECEIVE)		
		BYTE PUBLIC;		
19	1	\$IF SW1		
		DECLARE LAST\$KEY (8) BYTE PUBLIC;		
		\$ENDIF		
		\$IF SW2		
		DECLARE LAST\$KEY (2) BYTE PUBLIC;		
		\$ENDIF		
20	1	DECLARE SERIAL(16) BYTE PUBLIC;		
21	1	DECLARE DISPLAY\$RAM(7CFH) BYTE AT(1000H) AUXILIARY;		
22	1	DECLARE		
		PARAMETER\$ADDRESS BYTE AT(0000H) AUXILIARY,		
		COMMAND\$ADDRESS BYTE AT(0001H) AUXILIARY;		
23	1	DECLARE (		
		DISPLAY\$RAM\$POINTER,		
		RASTER,		
		LINEO,		
		L)		
		WORD PUBLIC;		



PL/M-51 COMPILER CRTCONTROLLER

RELOCATED RELOCATED

SEJECT

THUS

/\* PROCEDURE READER: THIS PROCEDURE IS WRITTEN IN ASSEMBLY LANGUAGE. THE  
EXTERNAL PROCEDURE SCANS THE 8 LINES OF THE KEYBOARD AND READS THE RETURN  
LINES. THE STATUS OF THE 8 RETURN LINES ARE THEN STORED IN INTERNAL  
MEMORY ARRAY CALLED CURRENTSKEY. THE PROCEDURE CONTROLS 2 STATUS FLAGS;  
KEY0 AND SAME. KEY0 IS SET IF ALL 8 SCANS READ NO KEY WAS PRESSED.  
IF ALL 8 SCANS ARE THE SAME AS THE LAST READING OF THE KEYBOARD, THEN  
SAME IS SET. \*/

24 2 READER: PROCEDURE EXTERNAL;  
25 1 END READER;

/\* PROCEDURE BLANK: THIS EXTERNAL PROCEDURE FILLS LINE0 SCAN WITH SPACES (20H ASCII)  
DURING THE SCROLL ROUTINES.\*/

26 2 BLANK: PROCEDURE EXTERNAL;  
27 1 END BLANK;

/\* PROCEDURE BLINE: THIS EXTERNAL PROCEDURE BLANKS FROM THE CURSER TO THE END OF  
THE DISPLAY LINE \*/

28 2 BLINE: PROCEDURE EXTERNAL;  
29 1 END BLINE;

/\* PROCEDURE FILL: THIS EXTERNAL PROCEDURE FILLS THE CURSER LINE  
WITH SPACES\*/

30 1 FILL:  
PROCEDURE EXTERNAL;  
31 1 END FILL;

PL/M-51 COMPILER CRTCONTROLLER

\$EJECT

```
/* PROCEDURE CHECK BAUD RATE: THIS PROCEDURE READS THE THREE PORT PINS ON P1 AND SETS UP
THE SERIAL PORT FOR THE SPECIFIED BAUD RATE */
```

```
32 1 CHECK$BAUD$RATE:
33 2 PROCEDURE;
34 2 SC0N=70H; /* MODE 1
35 2 /* ENABLE RECEPTION*/
36 2 /* TIMER 1 AUTO RELOAD */
37 2 /* TIMER 1 ON */
38 2 /* ENABLE SERIAL INTERRUPT*/
39 2 /* SERIAL INTERRUPT MASK FLAG */
40 3 /* 00 IS NOT ALLOWED */
41 3 TH1=040H; /* 150 BAUD */
42 3 TH1=0A0H; /* 300 BAUD */
43 3 TH1=0D0H; /* 600 BAUD */
44 3 TH1=0E8H; /* 1200 BAUD */
45 3 TH1=0F4H; /* 2400 BAUD */
46 3 TH1=0FAH; /* 4800 BAUD */
47 3 TH1=0FDH; /* 9600 BAUD */
48 1 END;
END CHECK$BAUD$RATE;
```

```
/* PROCEDURE LOAD CURSER: LOAD CURSER TAKES THE VALUE HELD IN RAM AND
LOADS IT INTO THE 8276 CURSER REGISTERS. */
```

```
49 1 LOAD$CURSER:
50 2 PROCEDURE;
51 2 IF CURSER$ON=1 THEN
52 2 CURSER$COL=CURSER$COLUMN; /* DISABLE BUFFER INTERRUPT */
53 2 COMMAND$ADDRESS=80H; /* INITIALIZE CURSER COMMAND */
54 2 PARAMETER$ADDRESS=CURSER$COL;
55 2 PARAMETER$ADDRESS=CURSER$ROW;
56 2 EX1=1; /* ENABLE BUFFER INTERRUPT */
57 1 END LOAD$CURSER;
```

```
/* PROCEDURE CARRIAGE$RETURN */
```

```
58 1 CARRIAGE$RETURN:
59 2 PROCEDURE;
60 2 DISPLAY$RAM$POINTER=DISPLAY$RAM$POINTER-CURSER$COLUMN;
61 2 CURSER$COLUMN=0;
62 2 CURSER$ON=1;
63 1 CALL LOAD$CURSER;
END CARRIAGE$RETURN;
```

\$EJECT

```

/* PROCEDURE DOWN CURSER: THIS PROCEDURE MOVES THE CURSER DOWN ONE ROW
BY ADDING 1 TO THE CURSER ROW RAM LOCATION THEN CALL LOAD CURSER */

```

```

64 1  DOWN$CURSER:
    PROCEDURE;
65 2  IF CURSER$ROW < 18H THEN
66 3  DO;
67 3      CURSER$ON=1;
68 3      CURSER$ROW=CURSER$ROW + 1;
69 3      CALL LOAD$CURSER;
70 3      IF DISPLAY$RAM$POINTER < 780H THEN
71 3          DISPLAY$RAM$POINTER=DISPLAY$RAM$POINTER + 50H;
72 3      ELSE
          DISPLAY$RAM$POINTER=(DISPLAY$RAM$POINTER-780H);
73 3      L=DISPLAY$RAM$POINTER-CURSER$COLUMN;
74 3      IF DISPLAY$RAM(L)=0F1H THEN
75 4      DO;
76 4          CALL FILL;
77 4          DISPLAY$RAM(L)=20H;
78 4      END;
79 3      END;
80 1  END DOWN$CURSER;

```

/\* LOOK FOR END OF \*/  
/\* LINE CHARACTER \*/  
/\* IF TRUE FILL LINE \*/  
/\* WITH SPACES \*/

```

/* PROCEDURE UP CURSER: THIS PROCEDURE MOVES THE CURSER UP ONE ROW
BY SUBTRACTING 1 TO THE CURSER ROW RAM LOCATION THEN CALL LOAD CURSER */

```

```

81 1  UP$CURSER:
    PROCEDURE;
82 2  IF CURSER$ROW > 0 THEN
83 3  DO;
84 3      CURSER$ROW=CURSER$ROW - 1;
85 3      CURSER$ON=1;
86 3      CALL LOAD$CURSER;
87 3      IF DISPLAY$RAM$POINTER < 50H THEN
88 3          DISPLAY$RAM$POINTER=DISPLAY$RAM$POINTER+780H;
89 3      ELSE
          DISPLAY$RAM$POINTER=DISPLAY$RAM$POINTER - 50H;
90 3      L=DISPLAY$RAM$POINTER-CURSER$COLUMN;
91 3      IF DISPLAY$RAM(L)=0F1H THEN
92 4      DO;
93 4          CALL FILL;
94 4          DISPLAY$RAM(L)=20H;
95 4      END;
96 3      END;
97 1  END UP$CURSER;

```

/\* LOOK FOR END OF LINE \*/  
/\* CHARACTER \*/  
/\* IF TRUE FILL WITH \*/  
/\* SPACES \*/

PL/M-51 COMPILER CRTCONTROLLER

BELLHOLMWOOD 12-14-71

```

SUBJECT                                TOPIC

/* PROCEDURE RIGHT CURSER: THIS PROCEDURE MOVES THE CURSER RIGHT ONE COLUMN
   BY ADDING 1 TO THE CURSER COLUMN RAM LOCATION THEN CALL LOAD CURSER */

98  1  RIGHT$CURSER:
      PROCEDURE;
99  2  IF CURSER$COLUMN < 4FH THEN
100 3  DO;
101 3      CURSER$COLUMN=CURSER$COLUMN + 1;
102 3      CURSER$ON=1;
103 3      CALL LOAD$CURSER;
104 3      DISPLAY$RAM$POINTER=DISPLAY$RAM$POINTER +1;
105 3  END;
106 1  END RIGHT$CURSER;

/* PROCEDURE LEFT CURSER: THIS PROCEDURE MOVES THE CURSER LEFT ONE COLUMN
   BY SUBTRACTING 1 TO THE CURSER COLUMN RAM LOCATION THEN CALL LOAD CURSER */

107 1  LEFT$CURSER:
      PROCEDURE;
108 2  IF CURSER$COLUMN > 0 THEN
109 3  DO;
110 3      CURSER$COLUMN=CURSER$COLUMN - 1;
111 3      CURSER$ON=1;
112 3      CALL LOAD$CURSER;
113 3      DISPLAY$RAM$POINTER=DISPLAY$RAM$POINTER -1;
114 3  END;
115 1  END LEFT$CURSER;

```



PL/M-51 COMPILER CR/CONTROLLER

SEJECT

TELE

/\* PROCEDURE MOV\$CURSER: THIS PROCEDURE IS USED IN CONJUNCTION WITH WORDSTAR  
IF A ESC F IS RECEIVED FROM THE HOST COMPUTER, THE TERMINAL CONTROLLER WILL  
READ THE NEXT TWO BYTE TO DETERMINE WHERE TO MOVE THE CURSER. THE FIRST BYTE  
IS THE ROW INFORMATION FOLLOWED BY THE COLUMN INFORMATION \*/

```

116 1  MOV$CURSER:
      PROCEDURE;
117 3  DO WHILE FIFO<4;          /* WAIT UNTILL THE MOV$CURSER PARAMETERS*/
118 3  END;                      /* ARE RECEIVED INTO THE FIFO */
119 2  TEMP=CURSER$ROW;
120 2  CURSER$ROW=SERIAL(2);
121 2  IF CURSER$ROW>TEMP THEN
122 3  DO;
123 4  L=DISPLAY$RAM$POINTER+ ((CURSER$ROW-TEMP)*50H);
124 4  IF L>7CFH THEN          /* IF OUT OF RAM RANGE */
125 4  DISPLAY$RAM$POINTER=L-7D0H; /* RAP AROUND TO BEGINNING */
126 4  ELSE                    /* OF RAM */
127 4  DISPLAY$RAM$POINTER=L;
128 2  ELSE
129 3  DO;
130 4  IF CURSER$ROW<TEMP THEN
131 4  L= (TEMP-CURSER$ROW)*50H;
132 4  IF DISPLAY$RAM$POINTER<L THEN
133 4  DISPLAY$RAM$POINTER=(7D0H-(L-DISPLAY$RAM$POINTER)); /* RAP AROUND TO END OF RAM */
134 4  ELSE
135 4  DISPLAY$RAM$POINTER=DISPLAY$RAM$POINTER-L;
136 3  END;
137 2  TEMP=CURSER$COLUMN;
138 2  CURSER$COLUMN=SERIAL(3);
139 2  IF CURSER$COLUMN>TEMP THEN
140 3  DISPLAY$RAM$POINTER=DISPLAY$RAM$POINTER+ (CURSER$COLUMN-TEMP);
141 2  ELSE
142 3  DISPLAY$RAM$POINTER=DISPLAY$RAM$POINTER- (TEMP-CURSER$COLUMN);
143 2  CURSER$ON=1;
144 2  CALL LOAD$CURSER;
145 2  L=DISPLAY$RAM$POINTER-CURSER$COLUMN;
146 2  IF DISPLAY$RAM(L)=0F1H THEN /* LOOK FOR END FO LINE CHARACTER*/
147 3  DO;
148 4  CALL FILL;              /* IF TRUE FILL WITH SPACES */
149 4  DISPLAY$RAM(L)=20H;
150 3  END;
151 2  ES=0;
152 2  DO I=2 TO FIFO-2;
153 3  SERIAL(I)=SERIAL(I+2);
154 2  END;
155 2  FIFO=FIFO-2;
156 2  ES=ENSP;
157 1  END MOV$CURSER;

```

PL/M-51 COMPILER CRTCONTROLLER

```

SELECT
/* PROCEDURE ERASE FROM CURSER TO END OF SCREEN: */

157 1 ERASE$FROM$CURSER$TO$END$OF$SCREEN:
PROCEDURE;
158 2 CALL BLINE; /* ERASE CURRENT LINE */
159 2 IF CURSER$ROW < 18H THEN
160 3 DO;
161 3 L=DISPLAY$RAM$POINTER-CURSER$COLUMN+50H; /* GET NEXT LINE */
162 4 DO WHILE (L < 7D0H) AND (L <> (LINE0 AND 7FFH));
163 4 DISPLAY$RAM(L)=0F1H; /* ERASE UNTIL LINE0 OR */
164 4 L=L+50H; /* END OF DISPLAY RAM */
165 4 END;
166 3 L=0;
167 4 DO WHILE L <> (LINE0 AND 7FFH); /* ERASE UNTIL LINE0 */
168 4 DISPLAY$RAM(L)=0F1H;
169 4 L=L+50H;
170 4 END;
171 3 END;
172 1 END ERASE$FROM$CURSER$TO$END$OF$SCREEN;

/* PROCEDURE HOME: THIS PROCEDURE MOVES THE CURSER TO THE 0,0 POSITION */

173 1 HOME:
PROCEDURE;
174 2 CURSER$ROW=00;
175 2 CURSER$COLUMN=00;
176 2 CURSER$ON=1;
177 2 CALL LOAD$CURSER;
178 2 DISPLAY$RAM$POINTER=(LINE0 AND 7FFH);
179 1 END HOME;

```

PL/M-51 COMPILER CRT/CONTROLLER

PL/M-51 COMPILER CRT/CONTROLLER

```

SELECT
/* PROCEDURE CLEAR SCREEN */
180 1 CLEAR$SCREEN:
PROCEDURE;
181 2 CALL HOME;
182 2 CALL ERASE$FROM$CURSER$TO$END$OF$SCREEN;
183 1 END CLEAR$SCREEN;

/* PROCEDURE SCROLL */
184 1 SCROLL:
PROCEDURE;
185 2 CALL BLANK;
186 2 EX0=0; /* DISABLE VERTICAL REFRESH INTERRUPT */
187 2 IF LINE0= 1F80H THEN
188 2 LINE0= 1800H;
189 2 ELSE
LINE0= LINE0+50H;
190 2 EX0=1; /* ENABLE VERTICAL REFRESH INTERRUPT */
191 1 END SCROLL;

/* PROCEDURE LINE$FEED */
192 1 LINE$FEED:
PROCEDURE;
193 2 IF CURSER$ROW=18H THEN
194 2 CALL SCROLL;
195 2 ELSE
DO;
CURSER$ROW= CURSER$ROW+1;
CURSER$ON=1;
CALL LOAD$CURSER;
END;
200 2 IF DISPLAY$RAM$POINTER > 77FH THEN
201 2 DISPLAY$RAM$POINTER=DISPLAY$RAM$POINTER-780H;
202 2 ELSE
DISPLAY$RAM$POINTER=DISPLAY$RAM$POINTER+50H;
203 2 L=DISPLAY$RAM$POINTER-CURSER$COLUMN;
204 2 IF DISPLAY$RAM(L)=0F1H THEN /* LOOK FOR END OF LINE CHARACTER*/
205 2 DO;
CALL FILL; /* IF TRUE FILL WITH SPACES */
206 2 DISPLAY$RAM(L)=20H;
207 2 END;
208 2 END LINE$FEED;
209 1

```

PL/M-51 COMPILER CRTCONTROLLER

\$EJECT

```

/* PROCEDURE DISPLAY: THIS PROCEDURE WILL TAKE THE BYTE IN RAM LABELED
RECEIVE AND PUT IT INTO THE DISPLAY RAM. */

```

```

210 1  DISPLAY:
211 2  PROCEDURE;
212 2  DISPLAY$RAM(DISPLAY$RAM$POINTER)=RECEIVE;
213 2  IF DISPLAY$RAM$POINTER=7CFH THEN /* IF END OF RAM */
214 2  DISPLAY$RAM$POINTER=000H; /* RAP AROUND TO BEGINNING */
215 2  ELSE
216 2  DISPLAY$RAM$POINTER=DISPLAY$RAM$POINTER+1;
217 2  IF CURSER$COLUMN=4FH THEN
218 3  DO;
219 3  CURSER$COLUMN=00H;
220 3  L=DISPLAY$RAM$POINTER;
221 3  IF DISPLAY$RAM(L)=0F1H THEN
222 4  DO;
223 4  CALL FILL;
224 4  DISPLAY$RAM(L)=20H;
225 4  END;
226 3  IF CURSER$ROW=18H THEN
227 3  CALL SCROLL;
228 3  ELSE
229 3  CURSER$ROW=CURSER$ROW+1;
230 3  END;
231 2  ELSE
232 2  CURSER$COLUMN=CURSER$COLUMN+1;
233 2  CURSER$QN=1;
234 2  CALL LOADCURSER;
235 1  END DISPLAY;

```



PL/M-51 COMPILER CRTCONTROLLER

PL/M-51 COMPILER CRTCONTROLLER

SELECT

TCHSE

/\* PROCEDURE DECIPHER: THIS PROCEDURE DECODES THE HOST COMPUTER'S MESSAGES AND DETERMINES WHETHER IT IS A DISPLAYABLE CHARACTER, CONTROL SEQUENCE, OR AN ESCAPE SEQUENCE THE PROCEDURE THEN ACTS ACCORDINGLY \*/

```

232 1  DECIPHER:
233 2  PROCEDURE;
234 2  START$DECIPHER:
235 3  VALID$RECEPTION=0;
236 3  I=0;
237 3  DO WHILE (I<FIFO) AND (SERIAL(I)>1FH) AND (SERIAL(I)<7FH);
238 3  RECEIVE=SERIAL(I);
239 3  CALL DISPLAY;
240 2  I=I+1;
241 3  END;
242 3  IF I>0 THEN
243 3  DO;
244 3  ES=0;
245 4  K=FIFO-I;
246 4  DO J=0 TO K;
247 4  SERIAL(J)=SERIAL(I);
248 4  I=I+1;
249 4  END;
250 3  FIFO=K;
251 3  ES=ENSP;
252 3  VALID$RECEPTION=1;
253 3  END;
254 3  IF FIFO=0 THEN
255 3  DO;
256 3  SERIAL$INT=0;
257 3  GOTO END$DECIPHER;
258 3  END;
259 3  IF (SERIAL(0)=1BH) THEN
260 3  DO;
261 3  IF (ESC$SEQ=1) AND (FIFO<2) THEN
262 3  GOTO END$DECIPHER;
263 3  K=(SERIAL(1) AND 5FH)-40H;
264 3  IF (K > 00H) AND (K<0CH) THEN
265 4  DO;
266 5  DO CASE K;
267 5  ;
268 5  CALL UP$CURSER;
269 5  CALL DOWN$CURSER;
270 5  CALL RIGHT$CURSER;
271 5  CALL LEFT$CURSER;
272 5  CALL CLEAR$SCREEN;
273 5  CALL MOV$CURSER;
274 5  ;
275 5  CALL HOME;
276 5  ;
277 5  CALL ERASE$FROM$CURSER$TO$END$OF$SCREEN;
278 5  CALL BLINK;
279 5  END;
280 4  ES=0;
281 3  /* DISABLE SERIAL INTERRUPTS WHILE MOVING FIFO */

```

PL/M-51 COMPILER CRT/CONTROLLER

```

280 4      DO I=0 TO (FIFO-2);
281 4          SERIAL(I)=SERIAL(I+2); /* MOVE FIFO */
282 4      END;
283 3      FIFO=FIFO-2;
284 3      ES=ENSP; /* ENABLE SERIAL INTERRUPTS */
285 3      VALID$RECEPTION=1;
286 3      IF FIFO=0 THEN
287 4          DO;
288 4              SERIAL$INT=0;
289 4              GOTO END$DECIPHER;
290 4          END;
291 3      END;
292 2      IF (SERIAL(0) > 07H) AND (SERIAL(0) < 0EH) THEN
293 3      DO;
294 4          DO CASE (SERIAL(0) - 08H);
295 4              CALL LEFT$CURSER; /* CTL H */
296 4              ;
297 4              CALL LINE$FEED; /* CTL J */
298 4              ;
299 4              CALL CLEAR$SCREEN; /* CTL L */
300 4              CALL CARRIAGE$RETURN; /* CTL M */
301 4          END;
302 3          ES=0; /* DISABLE SERIAL INTERRUPTS WHILE MOVING FIFO */
303 4          DO I=0 TO (FIFO-1);
304 4              SERIAL(I)=SERIAL(I+1); /* MOVE FIFO */
305 4          END;
306 3          FIFO=FIFO-1;
307 3          ES=ENSP; /* ENABLE SERIAL INTERRUPTS */
308 3          VALID$RECEPTION=1;
309 3      END;
310 2      IF VALID$RECEPTION=0 THEN
311 3      DO;
312 3          ES=0;
313 4          DO I=0 TO (FIFO-1); /* IF CHARACTER IS UNRECOGNIZED THEN */
314 4              SERIAL(I)=SERIAL(I+1); /* TRASH IT */
315 4          END;
316 3          FIFO=FIFO-1;
317 3          ES=ENSP;
318 3      END;
319 2      IF FIFO=0 THEN
320 2          SERIAL$INT=0;
321 2      END$DECIPHER;
321 2      END DECIPHER;

```

\$EJECT

/\* PROCEDURE TRANSMIT- THIS PROCEDURE LOOKS AT THE CLEAR TO SEND PIN FOR AN ACTIVE LOW SIGNAL. ONCE THE MAIN COMPUTER SIGNALS THE 8051 THE ASCII CHARACTER IS PUT INTO THE SERIAL PORT.\*/

```

322 1  TRANSMIT:
      PROCEDURE;
323 2  IF LOCAL$LINE =1 THEN
324 3  DO;
325 4      DO WHILE (CLEAR$TO$SEND=1) OR (TRANSMIT$INT=0);
326 4      END;
327 3      SEUF=ASCII$KEY;
328 3      TRANSMIT$INT=0;
329 3  END;
330 2  ELSE
      DO;
331 3      SERIAL(FIFO)=ASCII$KEY;
332 3      FIFO=FIFO+1;
333 3      SERIAL$INT=1;
334 3  END;
335 1  END TRANSMIT;

```

/\* PROCEDURE AUTO\$REPEAT: THIS PROCEDURE WILL PERFORM AN AUTO REPEAT FUNCTION AFTER A FIXED DELAY PERIOD \*/

```

336 1  AUTO$REPEAT:
      PROCEDURE;
337 2  IF NEW$KEY=1 THEN
338 3  DO;
339 3      TRANSMIT$TOGGLE=0;
340 3      TRANSMIT$COUNT=00H;
341 3      CALL TRANSMIT;          /* FIRST CHARACTER */
342 3      NEW$KEY=0;
343 3  END;
344 2  ELSE
      DO;
345 3      IF TRANSMIT$COUNT <> 00H THEN
346 4      DO;
347 4          TRANSMIT$COUNT=TRANSMIT$COUNT+1;
348 4          IF TRANSMIT$COUNT=0FFH THEN /*DELAY BETWEEN FIRST CHARACTER AND THE SECOND */
349 5          DO;
350 5              CALL TRANSMIT;          /*SECOND CHARACTER */
351 5              TRANSMIT$COUNT=00;
352 5          END;
353 4      END;
354 3      ELSE
          DO;
355 4          CURSER$QN=1;
356 4          CURSER$COUNT=0;
357 4          IF TRANSMIT$TOGGLE = 1 THEN /* 2 VERT FRAMES BETWEEN 3RD TO NTH CHARACTER */
358 4          CALL TRANSMIT;          /* 3RD THROUGH NTH CHARACTER */
359 4          TRANSMIT$TOGGLE= NOT TRANSMIT$TOGGLE;
360 4      END;
361 3  END;
362 1  END AUTO$REPEAT;

```

PL/M-51 COMPILER CRTCONTROLLER

```

SUBJECT: ***** START MAIN PROGRAM *****/
/* BEGIN BY PUTTING ASCII CODE FOR BLANK IN THE DISPLAY RAM;*/

363 1 INIT:
DO L=0 TO 7Fh;
364 2 DISPLAY$RAM(L)=20h;
365 2 END;

/* INITIALIZE POINTERS, RAM BITS, ETC. */

366 1 ESC$SEQ=0;
367 1 SCAN$INT=0;
368 1 SERIAL$INT=0;
369 1 FIFO=0;
370 1 CURSER$COUNT=0;
371 1 LLC=0;
372 1 DATA$TERMINAL$READY=1;
373 1 TC0N=05h;
374 1 LINE0=1800h;
375 1 RASTER=1800h;
376 1 DISPLAY$RAM$POINTER=0000h;
377 1 TRANSMIT$INT=1;

$IF SW1
DO I=0 TO 7;
378 2 LAST$KEY(I)=00h;
379 2
380 2 END;

381 1 VALID$KEY=0;
382 1 LAST$SHIFT$KEY=1;
383 1 LAST$CONTROL$KEY=1;
384 1 LAST$CAP$LOCK=1;
$ENDIF

$IF SW2
RCVFLG=0;
SYNC=0;
BYFIN=0;
KBDINT=0;
ERROR=0;
$ENDIF

/* INITIALIZE THE 8276 */

385 1 COMMAND$ADDRESS=00h; /* RESET THE 8276 */
386 1 PARAMETER$ADDRESS=4Fh; /* NORMAL ROWS, 80 CHARACTER/ROW */
387 1 PARAMETER$ADDRESS=58h; /* 2 ROW COUNTS PER VERTICAL RETRACE
25 ROWS PER FRAME */
388 1 PARAMETER$ADDRESS=89h; /* LINE 9 IS THE UNDERLINE POSITION
10 LINES PER ROW */
389 1 PARAMETER$ADDRESS=0F9h; /* OFFSET LINE COUNTER, NON-TRANSPARENT FIELD ATTRIBUTE

```



PL/M-51 COMPILER CRTCONTROLLER

RELIGHTMODE 32-4-1-1

NON-BLINKING UNDERLINE CURSER, 20 CHARACTER COUNTS PER  
HORIZONTAL RETRACE \*/

```

390 1  TEMP=COMMAND$ADDRESS; *****
391 1  CURSER$COLUMN=00H;
392 1  CURSER$ROW=00H;
393 1  CURSER$COL=00H;
394 1  CALL LOAD$CURSER;
395 1  TEMP=COMMAND$ADDRESS;

396 1  COMMAND$ADDRESS=0E0H; /* PRESET 8276 COUNTERS */
397 1  TEMP=COMMAND$ADDRESS;

398 1  COMMAND$ADDRESS=23H; /* START DISPLAY */
399 1  COMMAND$ADDRESS=0A0H; /* ENABLE INTERRUPTS */
400 1  TEMP=COMMAND$ADDRESS;

/* SET UP INTERRUPTS AND PRIORITIES */

$IF SW1
401 1  IP=10H; /* SERIAL PORT HAS HIGHEST PRIORITY */
402 1  IE=85H; /* ENABLE 8051 EXTERNAL INTERRUPTS */
$ENDIF

$IF SW2
IP=10H; /* SERIAL PORT HAS HIGHEST PRIORITY */
IE=87H; /* ENABLE TIMER0 INTERRUPT */
TMOD=05H; /* TIMER 0 =EVENT COUNTER */
TL0=0FFH;
TH0=0FFH; /* INITIALIZE COUNTER TO FFFFH */
TR0=1;
$ENDIF

/* PROCEDURE SCANNER: THIS PROCEDURE SCANS THE KEYBOARD AND DETERMINES IF A
SINGLE VALID KEY HAS BEEN PUSHED. IF TRUE THEN THE ASCII EQUIVALENT
WILL BE TRANSMITTED TO THE HOST COMPUTER.*/

403 1  SCANNER:
EA=1;
404 1  DATA$TERMINAL$READY=0;
405 1  IF CURSER$COUNT=1FH THEN /* PROGRAMMABLE CURSER BLINK */
406 2  DO;
407 2  CURSER$ON=NOT CURSER$ON;
408 2  CURSER$COUNT=00;
409 2  IF CURSER$ON=0 THEN
410 2  CURSER$COL=7FH;
411 2  CALL LOAD$CURSER;
412 2  END;
413 1  IF LLC<>LOCAL$LINE THEN /* IF LOCAL/LINE HAS CHANGED STATUS */
414 2  DO;
415 2  IF LOCAL$LINE=0 THEN
416 3  DO;
417 3  ENSP=0;
418 3  ES=0;
419 3  END;
420 2  ELSE
421 2  CALL CHECK$BAUD$RATE;
422 2  LLC=LOCAL$LINE;
423 2  $IF SW1
424 2  DO WHILE SCAN$INT=0; /* WAIT UNTIL VERTICAL RETRACE BEFORE */
425 2  IF SERIAL$INT=1 THEN /* SCANNING THE KEYBOARD */
426 2  CALL DECIPHER;
END;

```

PL/M-51 COMPILER CRIOCONTROLLER

RELOCATED CRIOCONTROLLER

```

$EJECT
427 1 CALL READER;
428 1 IF VALID$KEY =1 AND SAME=1 AND (LAST$SHIFT$KEY=SHIFT$KEY) AND
      (LAST$CAP$LOCK=CAP$LOCK) AND (LAST$CONTROL$KEY=CONTROL$KEY) THEN
429 1 CALL AUTO$REPEAT;
430 1 ELSE
      DO;
431 2 IF KEY0=0 AND SAME=0 THEN
432 3 DO;
433 3 TEMP =0;
434 3 K=0;
435 4 DO WHILE LAST$KEY (K)=0;
436 4 K=K+1;
437 4 END;
438 3 TEMP=LAST$KEY (K);
439 4 DO I=(K+1) TO 7;
440 4 TEMP=TEMP+LAST$KEY (I);
441 4 END;
442 3 IF TEMP=LAST$KEY (K) THEN
443 4 DO;
444 4 J=0;
445 5 DO WHILE (TEMP AND 01H)=0;
446 5 TEMP=SHR (TEMP,1);
447 5 J=J+1;
448 5 END;
449 4 IF TEMP >1 THEN
450 5 DO;
451 5 VALID$KEY=0;
452 5 NEW$KEY=0;
453 5 END;
454 4 ELSE
455 5 DO;
456 5 IF CONTROL$KEY=0 THEN
457 5 ASCII$KEY=(LOW$SCAN (K).KEY (J)) AND 1FH;
458 6 IF SHIFT$KEY=0 THEN
459 6 ASCII$KEY=LOW$SCAN (K+08H).KEY (J);
460 6 ELSE
461 7 DO;
462 7 ASCII$KEY=LOW$SCAN (K).KEY (J);
463 7 IF (CAP$LOCK=0) AND (ASCII$KEY>60H) AND (ASCII$KEY<7BH) THEN
464 7 ASCII$KEY=ASCII$KEY-20H;
465 8 IF LLC=0 THEN
466 8 DO;
467 8 IF ASCII$KEY=1BH THEN
468 8 ESC$SEQ=1;
469 8 ELSE
470 8 ESC$SEQ=0;
471 8 END;
472 7 END;
473 5 LAST$SHIFT$KEY=SHIFT$KEY;
474 5 LAST$CAP$LOCK=CAP$LOCK;
475 5 LAST$CONTROL$KEY=CONTROL$KEY;
476 5 VALID$KEY=1;
477 5 NEW$KEY=1;
478 4 END;
479 3 ELSE
480 4 DO;
481 4 VALID$KEY=0;
482 4 NEW$KEY=0;
483 3 END;
484 2 END;
$ENDIF

```

\$EJECT

TABLE

```

$IF SW2
IF SERIAL$INT=1 THEN
  CALL DECIPHER;
IF KBDINT =1 THEN
DO;
  IF ERROR =0 THEN
  DO;
    ASCII$KEY=LS$KEY(1);
    NEWS$KEY=1;
    CALL AUTO$REPEAT;
    KBDINT=0;
  END;
  ERROR=0;
  KBDINT=0;
END;
$ENDIF

```

```

485 1 GOTO SCANNER;
486 1 END;

```

# MODULE INFORMATION:

```

CODE SIZE
CONSTANT SIZE
DIRECT VARIABLE SIZE
INDIRECT VARIABLE SIZE
BIT SIZE
BIT-ADDRESSABLE SIZE
AUXILIARY VARIABLE SIZE
MAXIMUM STACK SIZE
REGISTER-BANK(S) USED:
1056 LINES READ
0 PROGRAM ERROR(S)
END OF PL/M-51 COMPILATION

```

## (STATIC+OVERLAYABLE)

```

= 08E6H 2278D
= 0080H 128D
= 2DH+00H 45D+ 0D
= 00H+00H 0D+ 0D
= 10H+00H 16D+ 0D
= 00H+00H 0D+ 0D
= 0000H 0D
= 0000H 12D
0

```

MCS-51 MACRO ASSEMBLER CRTASM

HEATING 03/05/82 12:00

ISIS-11 MCS-51 MACRO ASSEMBLER V2.1  
 OBJECT MODULE PLACED IN :F1:CRTASM.OBJ  
 ASSEMBLER INVOKED BY: ASM51 :F1:CRTASM.SRC

LUC OBJ LINE SOURCE

```

1
2
3
4 ;
5 ;
6
7 PUBLIC BLANK
8 PUBLIC BLINE
9 PUBLIC FILL
10 EXTRN DATA (LINE0,MASTER,POINT,SERIAL,FIFO,CURSER,COUNT,L)
11 EXTRN BIT (SERINT,ESCSEQ,TRNINT,SCAN)
12
13
14 CSEG AT(03H)
15 SJMP VENT
16
17 ;
18 ;
19 ;
20
21 CSEG AT(013H)
22 SJMP BUFFER
23
24 CSEG AT(023H)
25 SJMP SERBUF
26
27 CSEG
28
29 VENT: PUSH PSH
30 PUSH ACC
31 PUSH U0M
32 MOV MASTER,LINE0
33 MOV MASTER+1,LINE0+1
34 MOV R0,#01H
35 MOVX A,R0
36 INC COUNT
37 SETB SCAN
38 POP U0M
39 POP ACC
40 POP PSH
41 RETI
42
43
44 BUFFER: PUSH PSH
45 PUSH ACL
46 PUSH UPL
47 PUSH LPM
48 ACALL DMA
49 POP UPM
50 POP UPL
51 POP ACL
52 POP PSH
53 RETI
54
55 +1 SEJECT

```





MCS-51 MACRO ASSEMBLER CNTASM

```

LUC OBJ      LINE SOURCE
56
0052 309904 57 SERBUF: JNB 099H,CVEN ;IF TRANSMIT BIT NOT SET THEN CHECK RECEIVE
0055 C299 58 CLR 099H ;CLR TRANSMISSION INTERRUPT FLAG
0057 0200 F 59 SETB 1RAINT ;SETB TRANS INT FOR PLM51 STATUS CHECK
0059 209028 60 OVER: JB 09H,GOBACK ;IF RI NOT SET GOBACK
005C C001 61 PUSH 01 ;PUSH REG USED BY PLM51
005E A999 62 MOV 01,09H ;READ SERBUF
0060 C298 63 CLR 098H ;CLEAR RI BIT
0062 C0D0 64 PUSH PSW ;PUSH REGS USED BY PLM51
0064 C0E0 65 PUSH ACC ;PUSH ACC
0066 C000 66 MOV 00H,00H ;PUSH 00H
0068 C200 F 67 SETCSEQ CLR 0ESCSEQ ;CLR ESC SEQUENCE FLAG
006A 7400 F 68 MOV A,#SERIAL ;GET SERIAL FIFO RAM START LOCATION
006C 2500 69 ADD A,#FIFC ;AND FIND HOW FAR INTO THE FIFO WE ARE
006E F8 70 MOV R0,A ;PUT IT INTO R0
006F E9 71 MOV A,R1 ;MOV R1 TO R0
0070 C2E7 72 CLR 0E7H ;CLR BIT 7 OF ACC 51
0072 F6 73 MOV 0E7H,A ;PUT DATA IN FIFO 51
0073 B41B02 74 CJNE A,#1BH,OVER1 ;IF DATA IS NOT A ESC KEY THEN GO OVER
0076 0200 F 75 SETB ESCSEQ ;SET ESC SEQUENCE FLAG
0078 0500 F 76 OVER1: INC FIFC ;MOV FIFC TO NEXT LOCATION
007A 0200 F 77 SETB SERINT ;SET SERIAL INT BIT FOR PLM51 STATUS CHECK
007C 0000 78 POP 00H ;POP REGISTERS
007E 00E0 79 POP ACC ;POP ACC
0080 0000 80 POP PSW ;POP PSW
0082 0001 81 POP 01H ;POP 01H
0084 32 82 GOBACK: RETI
83
0085 C000 84 BLANK: PUSH PSW ;PUSH REG USED BY PLM51
0087 C0E0 85 PUSH ACC ;PUSH ACC
0089 C082 86 PUSH DPL ;PUSH DPL
008B C083 87 PUSH DPH ;PUSH DPH
008D C000 88 MOV 00H,00H ;PUSH 00H
008F 850082 F 89 MOV 00H,LINE0+1 ;GET LINE0 INFO
0092 850083 F 90 MOV 00H,LINE0 ;AND PUT IT INTO DPH
0095 7850 91 MOV 00H,#50H ;NUMBER OF CHARACTERS IN A LINE
0097 7420 92 NOTYET: MOV A,#20H ;ASCII SPACE CHARACTER
0099 F01A00 93 MOVX A,CPTR,A ;MOV TO DISPLAY RAM
009A A330 94 INC DPTR ;INCR TO NEXT DISPLAY RAM LOCATION
009B 08FA 95 DJNZ R0,NOTYET ;IF ALL 50H LOCATIONS ARE NOT FILLED
96 ;GO DO MORE
009D 0000 97 POP 00H ;POP REGISTERS
009F 0083 98 POP 00H ;POP 00H
00A1 0082 99 POP DPL ;POP DPL
00A3 00E0 100 POP ACC ;POP ACC
00A5 0000 101 POP PSW ;POP PSW
00A7 22 102 RET
103 +1 SEJECT

```

MCS-51 MACRO ASSEMBLER CKTASM

LUC OBJ LINE SOURCE

```

104
00A8 00000000 105 BLINK: PUSH PSW ;PUSH REGISTERS USED BY PLM51
00AA 00E0 106 PUSH ACL
00AC 00D2 107 PUSH DPL
00AE 0083 108 PUSH DPH
00B0 0000 109 PUSH V0H
00B2 850083 F 110 MOV DPH,PCINI ;GET CURRENT DISPLAY RAM LOCATION
00B5 850082 F 111 MOV DPL,PCINT+1
00B8 438310 112 URL DPH,#10H ;SET BIT 15 FOR RAM ADDRESS DECODING
00BB 8800 F 113 MOV R0,CURSEK ;GET CURSER COLUMN INFO TO TELL HOW
;FAR INTO THE ROW YOU ARE
00BD 7420 114 CONT1: MOV R4,A,#20H ;ASCII SPACE CHARACTER
00BF F0 115 MOVX @DPTN,A ;MOV TO DISPLAY RAM
00C0 A3 116 INC DPTR ;INCR TO NEXT DISPLAY RAM LOCATION
00C1 08 117 INC R0
00C2 8800F8 118 CJNE R0,#50H,CONT1 ;IF NOT AT THE END OF THE LINE
;CONTINUE
00C5 0000 119 POP V0H ;POP REGISTERS
00C7 0083 120 POP DPH
00C9 0082 121 POP DPL
00CB 00E0 122 POP ACC
00CD 0000 123 POP PSW
00CF 22 124 NET
125
126
127
00D0 0000 128 FILL: PUSH PSW ;PUSH REGISTERS USED BY PLM51
00D2 00E0 129 PUSH ACC
00D4 0082 130 PUSH DPL
00D6 0083 131 PUSH DPH
00D8 0000 132 PUSH V0H
00DA C3 133 CLR C
00DB 850083 F 134 MOV DPH,L ;GET BEGINNING OF LINE RAM LOCATION
00DE 850082 F 135 MOV DPL,L+1 ;CALCULATED BY PLM51
00E1 438310 136 URL DPH,#10H ;SET BIT 15 FOR DISPLAY RAM ADDRESS DECODE
00E4 784F 137 MOV R0,#4FH ;SET UP COUNTER FOR 50H LOCATIONS
00E6 A3 138 INC DPTR ;GO PAST THE OFH
00E7 7420 139 CONT2: MOV R4,A,#20H ;ASCII SPACE CHARACTER
00E9 F0 140 MOVX @DPTN,A ;MOVE TO DISPLAY RAM
00EA A3 141 INC DPTR ;INCR TO NEXT DISPLAY RAM LOCATION
00EB 08FA 142 DJNZ R0,CUNT2 ;IF ALL 79 LOCATIONS HAVE NOT BEEN FILLED
;THEN CONTINUE
00ED 0000 143 POP V0H ;POP REGISTERS
00EF 0083 144 POP DPH
00F1 0082 145 POP DPL
00F3 00E0 146 POP ACL
00F5 0000 147 POP PSW
00F7 22 148 NET
149
150
151
152 +1 SEJECT

```

MCS-51 MACRO ASSEMBLER

CKTASH

CKTASH

MCS-51 MACRO ASSEMBLER

LUC	OBJ	LINE	SOURCE	OBJECT	HEX	DISP
		153				EA 0710
		154	*****			EA 0710
		155	;THIS ROUTINE MOVES DISPLAY RAM DATA TO ROW BUFFER OF 8276			EA 0710
		156	*****			EA 0710
		157				EA 0710
00F0	21dB	158	DDONE: AJMP	UMACNE		03 0710
		159				EA 0710
00FA	850083 F	160	UMA: MOV	UPH,MASTER		03 0710
00FD	850082 F	161	MOV	UPL,MASTER+1		EA 0710
0100	E0	162	MOVX	A,ADPTR		03 0710
0101	A3	163	INC	UPTR		EA 0710
0102	2003F3	164	JB	UB3F,DDONE		EA 0710
0103	E0	165	MOVX	A,ADPTR		EA 0710
0106	A3	166	INC	UPTR		EA 0710
0107	E0	167	MOVX	A,ADPTR		EA 0710
0108	A3	168	INC	UPTR		EA 0710
0109	E0	169	MOVX	A,ADPTR		EA 0710
010A	A3	170	INC	UPTR		EA 0710
010B	E0	171	MOVX	A,ADPTR		EA 0710
010C	A3	172	INC	UPTR		EA 0710
010D	E0	173	MOVX	A,ADPTR		EA 0710
010E	A3	174	INC	UPTR		EA 0710
010F	E0	175	MOVX	A,ADPTR		EA 0710
0110	A3	176	INC	UPTR		EA 0710
0111	E0	177	MOVX	A,ADPTR		EA 0710
0112	A3	178	INC	UPTR		EA 0710
0113	E0	179	MOVX	A,ADPTR		EA 0710
0114	A3	180	INC	UPTR		EA 0710
0115	E0	181	TEN: MOVX	A,ADPTR		EA 0710
0116	A3	182	INC	UPTR		EA 0710
0117	E0	183	MOVX	A,ADPTR		EA 0710
0118	A3	184	INC	UPTR		EA 0710
0119	E0	185	MOVX	A,ADPTR		EA 0710
011A	A3	186	INC	UPTR		EA 0710
011B	E0	187	MOVX	A,ADPTR		EA 0710
011C	A3	188	INC	UPTR		EA 0710
011D	E0	189	MOVX	A,ADPTR		EA 0710
011E	A3	190	INC	UPTR		EA 0710
011F	E0	191	MOVX	A,ADPTR		EA 0710
0120	A3	192	INC	UPTR		EA 0710
0121	E0	193	MOVX	A,ADPTR		EA 0710
0122	A3	194	INC	UPTR		EA 0710
0123	E0	195	MOVX	A,ADPTR		EA 0710
0124	A3	196	INC	UPTR		EA 0710
0125	E0	197	MOVX	A,ADPTR		EA 0710
0126	A3	198	INC	UPTR		EA 0710
0127	E0	199	MOVX	A,ADPTR		EA 0710
0128	A3	200	INC	UPTR		EA 0710
0129	E0	201	TWENTY: MOVX	A,ADPTR		EA 0710
012A	A3	202	INC	UPTR		EA 0710
012B	E0	203	MOVX	A,ADPTR		EA 0710
012C	A3	204	INC	UPTR		EA 0710
012D	E0	205	MOVX	A,ADPTR		EA 0710
012E	A3	206	INC	UPTR		EA 0710
012F	E0	207	MOVX	A,ADPTR		EA 0710



MCS-51 MACRO ASSEMBLER

CNTASK

CNTASK

MCS-51 MACRO ASSEMBLER

LOC	OBJ	LINE	SOURCE	LOC	OBJ	LINE	SOURCE
0130	A3	208	INC UPTR	0130	A3	208	INC UPTR
0131	E0	209	MOVX A,CDPTH	0131	E0	209	MOVX A,CDPTH
0132	A3	210	INC UPTR	0132	A3	210	INC UPTR
0133	E0	211	MOVX A,CDPTH	0133	E0	211	MOVX A,CDPTH
0134	A3	212	INC UPTR	0134	A3	212	INC UPTR
0135	E0	213	MOVX A,CDPTH	0135	E0	213	MOVX A,CDPTH
0136	A3	214	INC UPTR	0136	A3	214	INC UPTR
0137	E0	215	MOVX A,CDPTH	0137	E0	215	MOVX A,CDPTH
0138	A3	216	INC UPTR	0138	A3	216	INC UPTR
0139	E0	217	MOVX A,CDPTH	0139	E0	217	MOVX A,CDPTH
013A	A3	218	INC UPTR	013A	A3	218	INC UPTR
013B	E0	219	MOVX A,CDPTH	013B	E0	219	MOVX A,CDPTH
013C	A3	220	INC UPTR	013C	A3	220	INC UPTR
013D	E0	221	THIRTY: MOVX A,CDPTH	013D	E0	221	THIRTY: MOVX A,CDPTH
013E	A3	222	INC UPTR	013E	A3	222	INC UPTR
013F	E0	223	MOVX A,CDPTH	013F	E0	223	MOVX A,CDPTH
0140	A3	224	INC UPTR	0140	A3	224	INC UPTR
0141	E0	225	MOVX A,CDPTH	0141	E0	225	MOVX A,CDPTH
0142	A3	226	INC UPTR	0142	A3	226	INC UPTR
0143	E0	227	MOVX A,CDPTH	0143	E0	227	MOVX A,CDPTH
0144	A3	228	INC UPTR	0144	A3	228	INC UPTR
0145	E0	229	MOVX A,CDPTH	0145	E0	229	MOVX A,CDPTH
0146	A3	230	INC UPTR	0146	A3	230	INC UPTR
0147	E0	231	MOVX A,CDPTH	0147	E0	231	MOVX A,CDPTH
0148	A3	232	INC UPTR	0148	A3	232	INC UPTR
0149	E0	233	MOVX A,CDPTH	0149	E0	233	MOVX A,CDPTH
014A	A3	234	INC UPTR	014A	A3	234	INC UPTR
014B	E0	235	MOVX A,CDPTH	014B	E0	235	MOVX A,CDPTH
014C	A3	236	INC UPTR	014C	A3	236	INC UPTR
014D	E0	237	MOVX A,CDPTH	014D	E0	237	MOVX A,CDPTH
014E	A3	238	INC UPTR	014E	A3	238	INC UPTR
014F	E0	239	MOVX A,CDPTH	014F	E0	239	MOVX A,CDPTH
0150	A3	240	INC UPTR	0150	A3	240	INC UPTR
0151	E0	241	FOXTY: MOVX A,CDPTH	0151	E0	241	FOXTY: MOVX A,CDPTH
0152	A3	242	INC UPTR	0152	A3	242	INC UPTR
0153	E0	243	MOVX A,CDPTH	0153	E0	243	MOVX A,CDPTH
0154	A3	244	INC UPTR	0154	A3	244	INC UPTR
0155	E0	245	MOVX A,CDPTH	0155	E0	245	MOVX A,CDPTH
0156	A3	246	INC UPTR	0156	A3	246	INC UPTR
0157	E0	247	MOVX A,CDPTH	0157	E0	247	MOVX A,CDPTH
0158	A3	248	INC UPTR	0158	A3	248	INC UPTR
0159	E0	249	MOVX A,CDPTH	0159	E0	249	MOVX A,CDPTH
015A	A3	250	INC UPTR	015A	A3	250	INC UPTR
015B	E0	251	MOVX A,CDPTH	015B	E0	251	MOVX A,CDPTH
015C	A3	252	INC UPTR	015C	A3	252	INC UPTR
015D	E0	253	MOVX A,CDPTH	015D	E0	253	MOVX A,CDPTH
015E	A3	254	INC UPTR	015E	A3	254	INC UPTR
015F	E0	255	MOVX A,CDPTH	015F	E0	255	MOVX A,CDPTH
0160	A3	256	INC UPTR	0160	A3	256	INC UPTR
0161	E0	257	MOVX A,CDPTH	0161	E0	257	MOVX A,CDPTH
0162	A3	258	INC UPTR	0162	A3	258	INC UPTR
0163	E0	259	MOVX A,CDPTH	0163	E0	259	MOVX A,CDPTH
0164	A3	260	INC UPTR	0164	A3	260	INC UPTR
0165	E0	261	FIFTY: MOVX A,CDPTH	0165	E0	261	FIFTY: MOVX A,CDPTH
0166	A3	262	INC UPTR	0166	A3	262	INC UPTR

# AP-223

MCS-51 MACRO ASSEMBLER

CMTASM

HEATG

MCS-51 MACRO ASSEMBLER

LCC	UPJ	LINE	SOURCE	LINE	UPJ
0167	E0	263	MOVX A,A,CPTM	218	0167
0168	A3	264	INC UPTR	219	0168
0169	E0	265	MOVX A,A,CPTM	220	0169
016A	A3	266	INC UPTR	221	016A
016B	E0	267	MOVX A,A,CPTM	222	016B
016C	A3	268	INC UPTR	223	016C
016D	E0	269	MOVX A,A,CPTM	224	016D
016E	A3	270	INC UPTR	225	016E
016F	E0	271	MOVX A,A,CPTM	226	016F
0170	A3	272	INC UPTR	227	0170
0171	E0	273	MOVX A,A,CPTM	228	0171
0172	A3	274	INC UPTR	229	0172
0173	E0	275	MOVX A,A,CPTM	230	0173
0174	A3	276	INC UPTR	231	0174
0175	E0	277	MOVX A,A,CPTM	232	0175
0176	A3	278	INC UPTR	233	0176
0177	E0	279	MOVX A,A,CPTM	234	0177
0178	A3	280	INC UPTR	235	0178
0179	E0	281	MOVX A,A,CPTM	236	0179
017A	A3	282	INC UPTR	237	017A
017B	E0	283	MOVX A,A,CPTM	238	017B
017C	A3	284	INC UPTR	239	017C
017D	E0	285	MOVX A,A,CPTM	240	017D
017E	A3	286	INC UPTR	241	017E
017F	E0	287	MOVX A,A,CPTM	242	017F
0180	A3	288	INC UPTR	243	0180
0181	E0	289	MOVX A,A,CPTM	244	0181
0182	A3	290	INC UPTR	245	0182
0183	E0	291	MOVX A,A,CPTM	246	0183
0184	A3	292	INC UPTR	247	0184
0185	E0	293	MOVX A,A,CPTM	248	0185
0186	A3	294	INC UPTR	249	0186
0187	E0	295	MOVX A,A,CPTM	250	0187
0188	A3	296	INC UPTR	251	0188
0189	E0	297	MOVX A,A,CPTM	252	0189
018A	A3	298	INC UPTR	253	018A
018B	E0	299	MOVX A,A,CPTM	254	018B
018C	A3	300	INC UPTR	255	018C
018D	E0	301	MOVX A,A,CPTM	256	018D
018E	A3	302	INC UPTR	257	018E
018F	E0	303	MOVX A,A,CPTM	258	018F
0190	A3	304	INC UPTR	259	0190
0191	E0	305	MOVX A,A,CPTM	260	0191
0192	A3	306	INC UPTR	261	0192
0193	E0	307	MOVX A,A,CPTM	262	0193
0194	A3	308	INC UPTR	263	0194
0195	E0	309	MOVX A,A,CPTM	264	0195
0196	A3	310	INC UPTR	265	0196
0197	E0	311	MOVX A,A,CPTM	266	0197
0198	A3	312	INC UPTR	267	0198
0199	E0	313	MOVX A,A,CPTM	268	0199
019A	A3	314	INC UPTR	269	019A
019B	E0	315	MOVX A,A,CPTM	270	019B
019C	A3	316	INC UPTR	271	019C
019D	E0	317	MOVX A,A,CPTM	272	019D

**AP-223**

CKTASM.

```

;ADD 79 TO BUFFER POINTER
;TO GET TO NEXT DISPLAY LINE
;IN THE DISPLAY MEMORY

```

ISIS-II MCS-51 MACRO ASSEMBLER v2.1  
 OBJECT MODULE PLACED IN \$F1:KEYBD.OBJ  
 ASSEMBLER INVOKED BY: ASM51 \$F1:KEYBD.SRC

```

LUC OBJ LINE SOURCE
1
2
3
4 *****
5 *****
6 *****
7 *****
8 *****
9 *****
10 *****
11 *****
12 *****
13 *****
14 *****
15 *****
16 *****
17 *****
18 *****
19 *****
20 *****
21 *****
22 *****
23 *****
24 *****
25 *****
26 *****
27 *****
28 *****
29 *****
30 *****
31 *****
32 *****
33 *****
34 *****
35 *****
36 +1 *****

```



MCS-51 MACRO ASSEMBLER

KEYBD

KEYBD

MCS-51 MACRO ASSEMBLER

LUC OBJ LINE SOURCE

```

37
38 UNDECODED_KEYBOARD SEGMENT CODE
39 NSEG UNDECODED_KEYBOARD
40
41
42
43 HEADER: PUSH PSW ;PUSH REG USED BY PLM51
44 PUSH ACC
45 PUSH DPL
46 PUSH UPH
47 PUSH UOH
48 PUSH U1H
49 PUSH U2H
50 PUSH U3H
51 MOV DPTR,#10FFH ;INITIALIZE DPTR TO KEYBOARD
52 ;ADDRESS
53 MOV R1,#00H ;CLR ZERO COUNTER
54 MOV R0,#LSKEY ;GET KEYBOARD RAM POINTIN
55 MOV R3,#08H ;INITIALIZE LOOP COUNTER
56 CLR KEY0 ;INITIALIZE PLM51 STATUS BITS
57 SETB SAME ;SET SAME BIT
58 MORE: MOV R0,#02H ;MOV LAST KEYBOARD SCAN TO R0
59 CLR A
60 MOVC A,A+DPTH ;SCAN KEYBOARD
61 CPL A ;INVERT
62 JZ ZERO ;IF SCAN WAS ZERO GO INCREMENT ZERO COUNTER
63 CJNE A,R0,SAME ;COMPARE WITH LAST SCAN IF NOT THE SAME
64 ;THEN CLR SAME BIT AND WRITE NEW INFORMATION
65 ;TO RAM
66 SJMP EQUAL ;IF EQUAL JMP OVER INCR OF ZERO COUNTER
67 ZERO: INC R1 ;INCR ZERO COUNTER
68 CJNE A,R0,SAME ;IF NOT THE SAME
69 EQUAL: INC R0 ;STEP TO NEXT SCAN RAM LOCATION
70 INC DPH ;NEXT KEYBOARD ADDRESS
71 DJNZ R3,MORE ;IF LOOP COUNTER NOT 0, SCAN AGAIN
72 CJNE R1,#08H,BACK ;CHECK TO SEE IF ALL 8 SCANS WHERE 0
73 SETB KEY0 ;IF YES SET KEY0 BIT
74 CLM SAME ;CLR SAME BIT
75 BACK: POP U3H ;POP REGISTERS
76 POP U2H
77 POP U1H
78 POP UOH
79 POP UPH
80 POP DPL
81 POP ACC
82 POP PSW
83 RET
84
85 NTSAME: MOV R0,A ;IF SCAN WAS NOT THE SAME THEN PUT NEW
86 ;SCAN INFO INTO RAM
87 CLR SAME ;CLR SAME BIT
88 SJMP EQUAL ;GO DO MORE
89
90
91 END

```

MCS-51 MACRO ASSEMBLER KEYBD

## SYMBOL TABLE LISTING

NAME	TYPE	VALUE	ATTRIBUTES
ACC . . . . .	D AUCTION	00E0H	A
BACN . . . . .	C AUCTION	003AH	R
DPH . . . . .	D AUCTION	0085H	A
DPL . . . . .	D AUCTION	0082H	A
EQUAL . . . . .	C AUCTION	002EH	R
KEYU . . . . .	B AUCTION	-----	EXT
LSTKEY . . . . .	D AUCTION	-----	EXT
MORE . . . . .	C AUCTION	0010H	R
NISAME . . . . .	C AUCTION	0046H	R
PSW . . . . .	D AUCTION	000UH	A
READER . . . . .	C AUCTION	000UH	R
SAME . . . . .	B AUCTION	-----	EXT
UNDECODED_KEYBOARD	C SEG	005UH	REL=UNII
ZERO . . . . .	C AUCTION	0029H	R

REGISTER BANK(S) USED: 0

ASSEMBLY COMPLETE, NO ERRORS FOUND

```

LUC  OBJ  LINE                               SOURCE
1          ;
2          ;
3          ;
4          ;
5          ;
6          ;
7          ;
8          ;
9          ;
10         ;
11         ;
12         ;
13         ;
14         ;
15         PUBLIC DETACH
16         EXTRN DATA (LSTKEY)
17         EXTRN BIT (KBDINT)
18         ;
19         ;
20         ;
21         ;
22         ;
23         ; "DECODE" INTERRUPT ROUTINE FOR DECODED KEYBOARDS
24         ;
25         ;
26 +1      SEJECT

```

LUC UBJ LINE SOURCE

```

27
28 DECODED_KEYBOARD SEGMENT CODE
29 KSEG DECODED_KEYBOARD
30
31 DETACH: PUSH PSH ;PUSH REGISTERS
32 PUSH LPL ;USED BY PLM51
33 PUSH LPM ;
34 PUSH ACC ;
35 MOV DPTR,#00FFH ;ADDRESS FOR KEYBOARD
36 CLM A
37 MOV A,A+DPTX ;FETCH ASCII BYTE
38 MOV LSIKEY+1,A ;MOV TO MEMORY TO BE READ BY PLM51
39 SETB KBUINT ;LET PLM51 KNOW THERE IS A BYTE
40 MOV THU,#0FFH ;SET COUNTER TO FFFFH SO INTERRUPT
41 MOV TLU,#0FFH ;ON THE NEXT FALLING EDGE OF T0
42 POP ACC
43 POP DPM ;POP REGISTERS
44 POP DPL
45 POP PSH
46 RETI
47
48
49
50
51 END

```



MCS-51 MACRO ASSEMBLER

DeCODE

000000

00000000 00000000 00000000

SYMBOL TABLE LISTING

000000

000000

-----

NAME	TYPE	VALUE	ATTRIBUTE	SIZE	ADDRESS
ACC	D	00000000		1	00000000
DECODED_KEYBOARD	C	00000000	REL=UNII	1	00000000
DETACH	C	00000000	SEG=DECODED_KEYBOARD	1	00000000
DPH	D	00000000		1	00000000
DPL	D	00000000		1	00000000
KBDINT	E	00000000	EXT	1	00000000
LSTKEY	D	00000000	EXT	1	00000000
PSW	D	00000000		1	00000000
THO	D	00000000		1	00000000
TLO	D	00000000		1	00000000
REGISTER BANK(S) USED: 00000000					
ASSEMBLY COMPLETE, NO ERRORS FOUND					

MCS-51 MACRO ASSEMBLER DETACH

MCS-51 MACRO ASSEMBLER DETACH

ISIS-11 MCS-51 MACRO ASSEMBLER V2.1  
 OBJECT MODULE PLACED IN :F1:DETACH.OBJ  
 ASSEMBLER INVOKED BY: ASM51 :F1:DETACH.SRC

LUC OBJ LINE SOURCE

LUC OBJ LINE SOURCE

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

18

19

+1

SELECT

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

```

LOC   OBJ      LINE          SOURCE
00B4      20           ;
        21           ;
        22
        23         INPUT EQU TO
        24
        25
        26         PUBLIC DETACH
        27         EXTRN DATA (LSIKEY)
        28         EXTRN @11 (RCVPLG,SYNC,BYF1N)
        29         EXTRN BIT (KBDINI,ERRON)
        30
        31 *****
        32           ;
        33           ;
        34 *****
        35 *****
        36           ;
        37           ;
        38           ;
        39 *****
        40 *****
        41           ;
        42           ;
        43           ;
        44           ;
        45           ;
        46           ;
0000      47         START0             EQU            000H    ;LOW BYTE FOR 150 BAUD
00F4      48         START1             EQU            0FH     ;HIGH BYTE FOR 150 BAUD
0000      49         MESSAGE0           EQU            000H    ;LOW BYTE FOR 150 BAUD
00E8      50         MESSAGE1           EQU            0E8H    ;HIGH BYTE FOR 150 BAUD
        51 +1 $EJECT

```

## MCS-51 MACRO ASSEMBLER DETACH

LUC	OBJ	LINE	SOURCE
		52	
		53	;
		54	*****
		55	;
		56	*****
		57	;
		58	*****
		59	
		60	
		61	
		62	DETACHABLE_KEYBOARD SEGMENT CODE
		63	NSSEG DETACHABLE_KEYBOARD
		64	
		65	DETACH: PUSH PSW ;PUSH REGISTERS USED BY PLM51
		66	PUSH ACC
0000	C000	67	JB NCVFLG,VALID ;IF RECEIVE FLAG SET GET NEXT BIT
0002	C0E0	68	JB INPLI,NSI ;IF T0 IS A 1 THEN NOT A START BIT
0004	200013	69	SETB NCVFLG ;IF T0 IS 0 THEN IT A START BIT
0007	20B44A	70	MOV TMO,NSIANT1 ;SET TIMER TO INTERRUPT IN THE MIDDLE OF START BIT
000A	0200	71	MOV TMO,NSIANT0
000C	758CF4	72	MOV A,TMO
000F	758A00	73	CLM ;SET TIMER COUNTER TO TIMER MODE
0012	E589	74	MOV TMO,A
0014	C2E2	75	SJMP FINI ;GO BACK TO PROGRAM
0016	F509	76	
0018	602B	77	VALID: JB SYNC,NXTBIT ;CHECK IF VALID START BIT HAS BEEN SEEN
		78	JB INPLI,NSI ;IF NOT CHECK IF VALID START BIT
001A	200010	79	SETB SYNC ;IF YES SET SYNC
001D	20B434	80	MOV LSTKEY,#00H ;INIT LSTKEY
0020	0200	81	MOV TMO,NSIANT1
0022	750080	82	MOV TMO,NSIANT0
0025	758CE8	83	SJMP FINI ;SET TIMER FOR 1 BIT TIME
0028	758A00	84	
002B	8018	85	NXTBIT: MOV TMO,NSIANT1 ;AND GO BACK TO MAIN PROGRAM
		86	MOV TMO,NSIANT0
002D	758CE8	87	JB BYFIN,STUP ;SET TIMER FOR 1 BIT TIME
0030	758A00	88	MOV A,LSTKEY ;CHECK TO SEE IF ALL 8 BITS HAVE BEEN RECEIVED
0033	200014	89	MOV C,INPLI ;GET WORKING REGISTER
0036	E500	90	RRC A ;GET NEXT BIT FROM T1
0038	A2B4	91	MOV LSIKEY,A
003A	13	92	JNC FINI ;IF NO CARRY THEN NOT DONE
003B	F500	93	SETB BYFIN
003D	5006	94	CLM
003F	0200	95	MOV LSIKEY+1,A
0041	C2E7	96	POP ACC
0043	F500	97	POP PSW
0045	00E0	98	RET
0047	0000		
0049	32		



AP-223

## SYMBOL TABLE LISTING

NAME	TYPE	VALUE	ATTRIBUTES
ACC . . . . .	D ADDR	00E0H	A
BYFIN . . . . .	B ADDR	----	EXT
DETACH . . . . .	C ADDR	0000H	R PUB
DETACHABLE_KEYBOARD	C SEG	0068H	REL=UNIT
ENR . . . . .	C ADDR	0052H	R
ENRUR . . . . .	B ADDR	----	EXT
FINI . . . . .	C ADDR	0045H	R
INPUT . . . . .	B ADDR	00B0H.4	A
KBDINT . . . . .	B ADDR	----	EXT
LSTKEY . . . . .	D ADDR	----	EXT
MESSAGE0 . . . . .	NLMB	0000H	A
MESSAGE1 . . . . .	NLMB	00E8H	A
NXTBIT . . . . .	C ADDR	002DH	R
PSW . . . . .	D ADDR	0000H	A
RCVFLG . . . . .	B ADDR	----	EXT
RST . . . . .	C ADDR	0054H	R
STANT0 . . . . .	NLMB	0000H	A
STANT1 . . . . .	NLMB	00F4H	A
STOP . . . . .	C ADDR	004AH	R
SYNC . . . . .	B ADDR	----	EXT
TU . . . . .	B ADDR	00B0H.4	A
TH0 . . . . .	D ADDR	008CH	A
TLO . . . . .	D ADDR	008AH	A
TMOD . . . . .	D ADDR	0089H	A
VALID . . . . .	C ADDR	001AH	R

REGISTER BANK(S) USED: 0

ASSEMBLY COMPLETE, NO ERRORS FOUND









*Extensive I/O subsystems and a tailored instruction set allow a 16-bit microcontroller to set its sights on a widening range of industrial and computer (and telecom and consumer) applications.*

## Controller chip takes on many industrial, computer uses

With industrial and computer control applications increasing all the time—and telecommunications and consumer applications emerging—designers increasingly need microcontrollers whose performance extends beyond that of the conventional 8-bit architectures. Normally, control system designers must depend on expensive and complex multiple-chip microprocessors to achieve high performance. But now, a 16-bit single-chip controller offers a much better solution. Not only does the 8096 offer perhaps the most extensive input/output “services” of any microcontroller, it also provides an instruction set and addressing modes tuned for both fast control operations and high-speed arithmetic.

In industrial applications, the 8096 can be used for process control, robotics, numerical and motor control, and instrumentation. Figure 1 shows the chip in a typical closed-loop servo system of the type used in industrial applications. In computer applications performance is the key feature, and here the 8096 provides greater throughput in systems in which simple data structures—a single I/O bit—and relatively small memories are required. Typical applications are computer peripherals such as printers, plotters, Winchesters, and other hard-disk systems.

In the consumer end, moreover, the 8096 is ideally suited for automotive engine and other controls (see “Stopping a Car”) and sophisticated video games. Both applications need the speed, calculating power, and addressability of a 16-bit microcomputer. For telecommunications, the controller is intended for high-speed modems, PABXs, and central office switching systems.

In addition to the full 16-bit CPU, the 8096's basic

architecture includes an 8-kbyte ROM and a 232-byte RAM, which serves as a register file. To meet the wide needs of controller environments, the chip contains an eight-channel, 10-bit analog-to-digital converter, a full-duplex UART (universal asynchronous receiver-transmitter), two 16-bit timers, and a programmable pulse-width-modulated output.

Since a microcontroller must be able to interface with various types of transducers and sensors, the 8096 features built-in, extensive I/O facilities. These include an eight-level priority interrupt structure, full-duplex serial I/O, parallel I/O, a watchdog timer, analog inputs for the A/D converter, a pulse-width modulated output and a high-resolution pulse output. Each of these facilities is integrated not only physically but logically into the chip's structure by being tightly coupled to the CPU.

The inherently high performance of a CPU suffers if the controller spends too much time administering complex real-time I/O operations. The 8096's on-board I/O facilities solve this problem by permitting the CPU to devote more time to executing mathematics and control algorithms and less on I/O.

**Table 1. Memory allocations of the 8096**

Locations	Uses
0000-0017	On-chip I/O
0018-0019	Data register/stack pointer
001A-00FF	Data registers (230 bytes)
0100-1FFD	Off-chip expansion RAM/ROM/I/O
1FFE-1FFF	On-chip I/O
2000-200F	Internal ROM interrupt vectors
2010-207F	Reserved
2080-3FFF	Internal ROM user program space
4000-FFFF	Off-chip expansion RAM/ROM/I/O

Steve Wiseman, Product Marketing Manager

Steve Burton, Senior Engineer

John Katausky, Technical Marketing Manager

Intel Corp., 5000 W. Williams Field Rd., Chandler, Ariz. 85224

## 16-bit microcontroller

The instruction set handles signed and unsigned 16-bit multiplications and divisions. Both 8-bit bytes and 16-bit double words are supported, and even 32-bit double words are supported for a subset of the main instruction set. A full 64 kbytes of memory address space is usable.

### A flexible register structure

The 8096 instruction set directly supports 256 bytes of registers, which can be referenced as 128-word registers or as 64 double-word registers. These registers also appear—for memory reference in-

structions—as the first 256 bytes of the 64-kbyte RAM address space. This permits, for example, the use of a portion of the register space as the subroutine stack on smaller systems that do not have external expansion memory (Table 1).

The first 24 bytes of this register space are reserved for on-chip I/O addresses. I/O locations are memory-mapped and can be referenced directly as registers. The word register located at address 18H serves as the stack pointer. Such a large register space allows a programmer to keep his most frequently referenced scalar variables in registers.

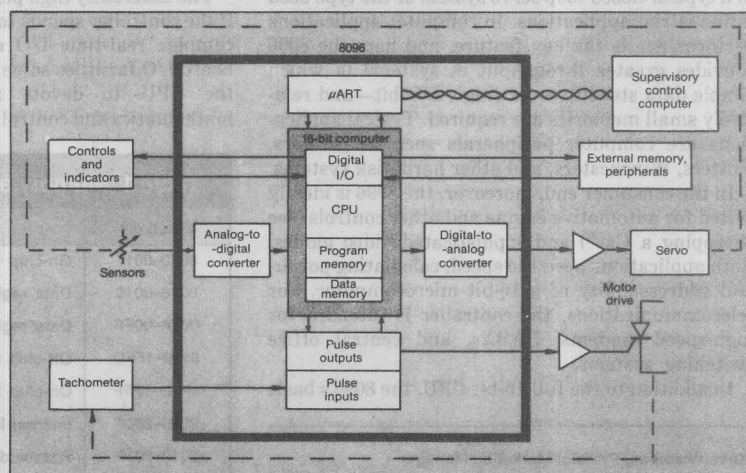
Table 2. Address modes of the 8096

Name	Time ( $\mu$ s) <sup>1</sup>	Time ( $\mu$ s) <sup>2</sup>	Written form	Action taken
Direct	0	N.A.	ADD B,A	$(B) \leq (B) + (A)$
Immediate	0.2 (word)	N.A.	ADDB B,#A	$(B) \leq (B) + A$
Immediate	0 (byte)	N.A.	ADD B,#A	$(B) \leq (B) + A$
Indirect	0.4	1.4	ADD B,(A)	$(B) \leq (B) + ((A))$
Autoincrement	0.6	1.6	ADD B,(A)+	$(B) \leq (B) + ((A)); (A) \leq (A) + \text{length of } A$
Short indexed	0.4	1.4	ADD B,C[A]	$(B) \leq (B) + ((A) + C); \text{ where } -128 < C < 127$
Short indexed	0.6	1.8	ST C[A],B	$((A) + C) \leq (B)$ ST, Pop only
Long indexed	0.6	1.6	ADD B[A],C	$(B) \leq (B) + ((A) + C)$
Long indexed	0.8	1.8	ST [A]C,B	$((A) + C) = (B)$ ST, Pop only

1. Addressed operand located in register space.

2. Addressed operand located in ROM space or external memory-expansion space.

Note: ST and Pop are the only instructions with an address-modified destination.



1. Sitting at the center of a closed-loop servo system, the 8096 microcontroller's I/O facilities interface with both digital and analog input signals. The 16-bit chip can handle virtually any type of computer or industrial control-system application.

Since the number of instructions required is reduced, fewer external memory references are needed. As a result, program execution is accelerated.

The 8096 uses separate internal instruction and data buses. With this architecture, ROM and external memory references are slightly slower than register references. Instructions cannot be executed out of the internal RAM register space, but external-expansion RAM instructions can be executed.

Both memory space and register space are fully byte-addressable. A 16-bit word begins on an even byte address, and the odd byte is the most significant

byte of the word. A 32-bit double word begins on an even-word address—both bit 0 and bit 1 of the address are zeros. Double words are produced by Multiply Words, Shift, and Normalize instructions and are used by Divide Words, Shift, and Normalize instructions. Double words are added and subtracted using Add with Carry and Subtract with Borrow instructions.

In most computers, the most commonly used instruction is Move. Because of the many registers in the 8096, a programmer can get away with fewer Move instructions—it is not necessary to switch

## Stopping a car

The 8096 can be very useful in an automotive antiskid braking system that allows a driver to decelerate his vehicle safely when one or more wheels start slipping. Slip conditions can be detected either by excessive wheel-speed differential or by excessive apparent deceleration, or both. For example, if one wheel hits an ice patch during heavy braking, the rotation of that wheel will slow down significantly, indicating a skid. On the other hand, normal tires begin to slip at or below 1 g of acceleration. If the apparent deceleration as measured by wheel speed is less than 1 g, the wheel is assumed to be skidding. Both skid-detection techniques depend on measuring a wheel's rotational speed.

Wheels are monitored by reluctance (magnetic) pickups, which generate pulse trains whose frequencies are proportional to wheel speed. Usually, a simple numerical relationship relates frequency to speed—13.3 Hz per miles per hour is a typical value. Four pickup outputs are easily handled by the 8096's high-speed input unit. At each transition of any of the pickups, the current value of timer 0 is saved in the input FIFO. The programmable edge detector in the high-speed input unit provides a convenient device for handling the wide dynamic range of the period measurement. At slow

speeds, the edge detector can be programmed to respond to both edges of the input signal; at medium speeds, to recognize only positive-going edges; and at high speeds, to respond to just one of eight positive-going edges. This technique not only extends the dynamic range of the measurement, but also reduces the interrupt overhead at high speeds.

Three successive time samples— $T_x$ ,  $T_y$ , and  $T_z$ —allow the speed and acceleration of a wheel to be determined from the following equations:

$$V_{xy} = C/(T_y - T_x) \quad (1)$$

$$V_{yz} = C/(T_z - T_y) \quad (2)$$

$$A_{xx} = (V_{yz} - V_{xy})/[0.5(T_z - T_x)] \quad (3)$$

where C is the reciprocal of the proportionality constant (e.g.,  $C = 1/13.3$ ),  $V_{xy}$ ,  $V_{yz}$ , and  $V_{zx}$  are velocities, and  $A_{xx}$  is acceleration.

The period of the incoming frequency is in units of 1.6  $\mu$ s since timer 0 is incremented at this rate. If  $I_x$  represents the value in the timer at time x and  $I_y$  the value at time y, then Eq. 1 and 2 are written as

$$V_{xy} = 46992.48/(I_y - I_x) \quad (4)$$

$$V_{yz} = 46992.48/(I_z - I_y) \quad (5)$$

In practice, the constants in Eq. 4 and 5 should be multiplied by a scaling factor to allow calculations to be performed in integer arithmetic. A factor of 100, for example, gives speed measured in units of 1/100 of a mile per hour.

Then the value 46992.48 becomes 4699248, which can easily be represented within 32 bits, and a 32-by-16-division instruction is used to perform the division.

Each wheel requires two such divisions—one for speed, one for acceleration—during each loop of the calculation. A typical loop takes about 10 ms. A typical 8-bit microprocessor takes 500 to 750  $\mu$ s per division, which means eight such divisions would require 4 to 6 ms. But the 8096 does all eight divisions in about 50  $\mu$ s. This speed improvement translates into a higher-performance module in response time or adaptability.

The watchdog timer of the 8096 helps enhance the reliability of the braking module. At a 15-MHz clock rate, the timer is incremented every 200 ns. During operation, the system software executes diagnostics periodically to ensure that the overall system—including hardware and software—is operating properly. If the operation is correct, the software will issue commands to reset the watchdog. But if a system failure prevents a diagnostic from running within a prescribed period, the watchdog timer will reset the entire system. The software cannot reset properly on an erroneous operation, such as a counter overflow, except by writing to the watchdog timer twice within its counting cycle.



## 16-bit microcontroller

operands in and out of memory locations. In addition, the chip's powerful three-operand instructions—Add, Subtract, Multiply and Logical And—often eliminate them. Since programmer productivity (measured in lines of code written per day) is reasonably constant, writing fewer Move instructions can lead to reduced development expense. Register Load and Store instructions, with a full set of addressing modes, handle moves that cannot be eliminated.

### Keeping addressing simple

Because a study of the ways in which addressing is used on the 8086 microprocessor indicates that programmers use complex addressing modes less than 0.7% of the time, the 8096's instruction set bypasses those in favor of the more commonly used addressing modes. But should complex addressing be needed, programmers can build such modes through macros. Addressing modes in the 8096 include direct, register-indirect, immediate, autoincrement, and both short (8-bit) and long (16-bit) indexed-address. Table 2 lists the address modes and the operations that occur when each is activated.

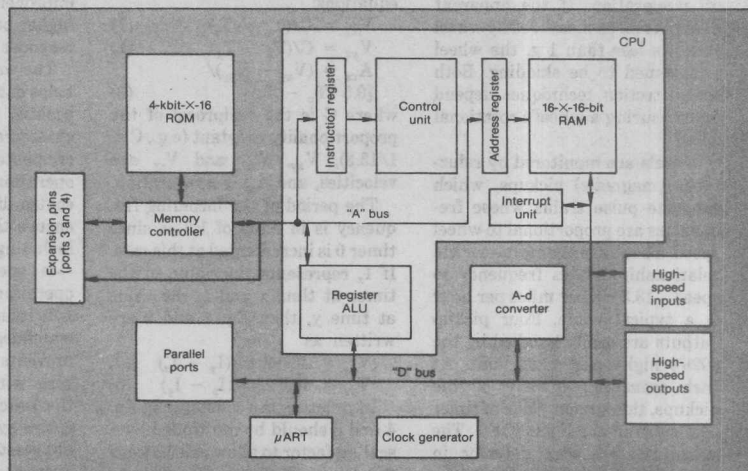
Indexed-address modes, by adding an 8- or 16-bit displacement to the contents of any 16-bit register

to form the effective address of an operand, allow fast access to arrays stored anywhere in memory. Indexed modes are also useful for referencing elements of based structures, as in the PL/M language. However, preliminary calculations are needed to reference a based array.

The stack pointer is fully addressable, as are all other 16-bit registers. As a result, it can be the base register for moded references. Stack-relative addressing, which is easy to program, is often used for recursive-subroutine parameter passing and dynamically allocated variables. While this technique does not make the best use of a large register space, it adds flexibility to the system. The stack need not be confined to internal RAM, but can fill any available RAM space in the system. The stack can flow across the boundary into register space at will, allowing recursion to very great subroutine depths.

Of the instruction set's 71 instructions, 25 take on both word and byte form, which increases the total to 96 instructions. The set includes 16 varieties of conditional jump, allowing for both signed and unsigned comparisons. All of the 2048 bits in register space can be tested individually by a Jump on Bit/Not Bit instruction. A Decrement and Jump on Not Zero instruction provides for loop control.

Since the number of instructions required is reduced, fewer external memory references are needed. As a result, program execution is accelerated.



2. Centered around the CPU and memory, the 8096's extensive I/O subsystems include an analog-to-digital converter, a universal asynchronous receiver-transmitter (UART), high-speed input and output circuitry, and a pulse-width modulation output circuit. Such intelligent I/O allows the CPU to concentrate not on real-time housekeeping but on high-speed arithmetic and control operations.

Most of the instructions execute in about 0.8  $\mu$ s; the longest, to normalize a zero, takes 8  $\mu$ s. All data-reference instructions except Pop, Push, and Normalize are available in byte form, and all such instructions except Jump on Bit and Normalize are available in word form. Table 3 lists some typical 8096 instructions and their run times.

A survey of code frequency usage shows that although most multiplications and divisions are unsigned, a signed form is still necessary. When unsigned multiplication and division instructions are preceded by a 0.8- $\mu$ s SIGND prefix, they are converted into full two's-complement signed multiplication and division. Either type of operation executes in less than 6  $\mu$ s. Word multiplications result in a double-word product, and byte multiplications produce a word product. With an instruction called Word Divide, a double-word dividend is divided by a word divisor to produce a word quotient and remainders.

Because jumps and calls are PC-relative, code is easy to relocate. Both Jump and Call instructions are available in a short 2-byte form with an 11-bit displacement. Jump on Bit is a 3-byte instruction with an 8-bit displacement. An indirect jump for the "do-case" is also provided.

In addition to the usual sign-extending (EXT) instructions for byte-to-word and word-to-double-word conversions, the set includes instructions LDBSA and LDBZE, which move a byte into a word with sign or zero extension. Most one- and two-operand forms execute in 1  $\mu$ s. Conditional jumps run in less than 1.8  $\mu$ s, and in about 0.8  $\mu$ s when the jump is not taken.

Shifts, whether by a specific number of bit positions or by a computed number, are provided for all three operand lengths (byte, word, and double word). In a floating-point software package, the mantissas must be aligned before they are added or subtracted, and the results normalized afterwards. Both functions require a software shift loop.

The Normalize instruction and the computer form of the Shift Double Word instruction allow fast software implementations of floating-point arithmetic with up to a 32-bit mantissa. Multibit shift instructions are very useful for scaling operations in scaled-integer arithmetic. Scaled-integer operations are usually faster than floating-point arithmetic in control applications.

In addition to an overflow flag, which is set by each arithmetic instruction, there is an overflow-trap flag. It can be checked at the end of a sequence of instructions to determine whether an overflow has occurred anywhere in the sequence.

The instruction set is complemented by a variety of I/O subsystems for handling virtually any com-

Table 3. Typical instruction times

Microseconds	Operands	Mnemonics	Descriptions
0.8	0	CLRC,SETC,DI,EI,CLRV,TSIGND	Flag manipulations
0.8	1	INC,DEC,CLR,NOT,NEG,SEX	One-operand instructions
0.8	2*	XOR,ADDC,SUB,AND,ADD,SUBC	Two-operand arithmetics
0.8	2*	OR,CMP	Two-op arithmetics
0.8	2*	LD,LDBSE,LDBZEST	Load and store registers
0.8 (not taken)	1	JC,JNC,ETC.	Conditional jumps
1.0	3*	AND,SUB,ADD	Three-op arithmetics
1.0 (not taken)	2	JBS,JBC	Jump on bit/ jump on not bit
1.6 (taken)	1	JC,JNC,ETC.	Conditional jumps
1.6	1	SJMP,IJMP,LJMP	Unconditional jumps
1.6 (stack register)	0	PUSHF	Push PSW
1.6 (stack register)	1*	PUSH	Stack push
1.8 (taken)	2	JBS,JBC,DJNZ	Jump on bit/ decrement and jump
1.6 + 0.2/shift	2	SHL,SHR,SHRA	Shift instructions
1.8	0	POPF	Pop PSW
2.2 + 0.2/shift	2	NORML	Normalize
2.4	1*	POP	Stack pop
2.4 (stack register)	1	LCALL,SCALL,RET	Subroutines
2.4 (stack external)	0	PUSHF	Push PSW
2.4 (stack external)	1*	PUSH	Stack push
2.6 (stack external)	0	POPF	Pop PSW
2.8 (stack external)	1*	POP	Stack pop
3.0 (stack external)	1	SCALL,CALL	Subroutines
3.2 (stack external)	0	RET	Subroutine
3.4	2*	MULB	Byte multiplication
3.6	2*	DIVB	Byte division
3.6	3*	MULB	Byte multiplication
5.2	2*	MUL	Word multiplication
5.2	2*	DIV	Word division
5.4	3*	MUL	Word multiplication

\*One of these operands may have full address modes.

## 16-bit microcontroller

puter peripheral or industrial application (Fig. 2). They include an a-d converter, a UART, timer-counters, and a programmable pulse-width-modulated output.

### I/O resources include a-d

The controller contains a complete eight-channel, 10-bit a-d converter. Using successive approximation to achieve high speed—33.6  $\mu$ s at a 15-MHz clock rate—it handles analog input voltages in the range of 0 to 5 V. An external reference is required and must be connected between the reference voltage and analog ground terminals. The converter generates a vectored interrupt when it completes a conversion cycle, allowing the CPU to have rapid access to the a-d input handler when operating in a multitask environment.

Conversion is initiated by writing to an 8-bit a-d command register. The results of a conversion are read from two 8-bit output data registers. One 8-bit register contains the eight most significant bits, and the other holds the two least significant bits, a 3-bit channel indicator, two unused bits, and a status bit. The status bit, which indicates whether the a-d conversion is still in progress, is typically used in a noninterrupt-driven environment.

Just four bits of the a-d command register are used. Three of the bits specify the channel to be converted, and the fourth specifies the method of initiating an a-d conversion cycle. For example, if the fourth bit is a 1, the cycle begins immediately after writing to the command register. If it is a 0,

the high-speed output logic subsystem initiates the conversion. The reason for the option is that many data acquisition algorithms require that conversions occur at specific intervals. This requirement is often difficult to manage through software because of interrupt latency and other conditions. Thus, the high-speed output subsystem provides the proper timing for periodic a-d conversions.

The 8096's UART is virtually a carbon copy of the one on the 8051 microcontroller. One of its 8-bit registers receives data, another transmits data, and another indicates the UART status plus bits to configure it for a specific operating mode. By setting the appropriate bits in the third, or control-status, register, a user can select one of four modes:

- Mode 0 (shift register) is a simple, synchronous mode in which the 8096 provides a clock to synchronize incoming or outgoing data. Mode 0 can also be used to expand the I/O.

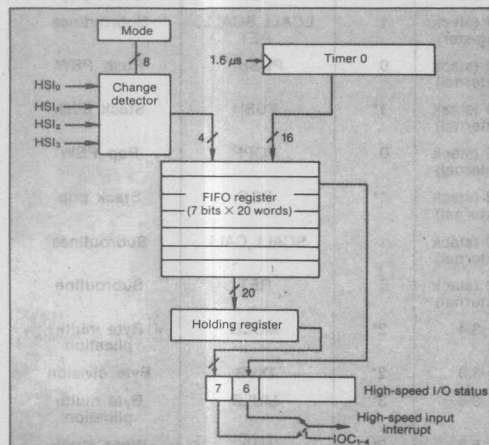
- Mode 1 is an 8-bit UART mode in which the eighth bit is used for parity when it is enabled.

- Mode 2 is a 9-bit UART mode in which the ninth bit is used for parity when it is enabled.

- Mode 3 is a 9-bit data/address mode in which the UART transmits and receives nine bits of data. This is useful for implementing a simple multiprocessor intercommunications link in which the ninth bit distinguishes address from data.

The remaining six bits of the control-status register are used for six operations: enabling the receiver section of the UART, enabling parity for both transmission and reception (even parity); storing the ninth bit when in the 9-bit transmitting mode; storing the ninth bit when in the 9-bit receiving mode, indicating that the receiver is ready, and indicating that the transmitter is ready. Also on board are a dedicated 15-bit baud-rate generator and a baud-rate clock that can be driven by either the 8096's crystal oscillator or an input at pin T2CLK. This gives maximum flexibility in setting baud rates.

The pulse-width modulated output can produce a pulse train of variable duty cycle, which can be integrated and clamped to provide an accurate digital-to-analog output function. The PWM circuit operates as follows: The 8096 crystal frequency is divided by three and clocks an 8-bit free-running counter. The counter output connects to one side of an 8-bit comparator; the other side of the comparator is tied to a user-addressable register. When the free-running counter value is the same as the one stored in the addressable register, an R-S flip-flop is set. The flip-flop is also reset when the counter rolls over from a count of 255 to 0. This produces a simple yet accurate variable duty-cycle oscillator, which can be programmed for a variable duty cycle from 0 to 255 in increments of X/256.



3. One half of the 8096's high-speed I/O subsystem is an input unit, which contains a user-programmable change detector that defines input transitions for the high-speed inputs. Each of the four inputs (HSI<sub>0</sub> - HSI<sub>3</sub>) can be programmed to respond to a different input transition.



## 16-bit microcontroller

The watchdog timer offers a simple way to recover from a software or hardware error. Essentially a 16-bit free-running counter that is clocked by the CPU clock generator circuitry, the timer is reset by writing a  $01E_H$  followed by a  $0E1_H$  to byte location 000AH. If a resetting does not occur at least once every 13.107 ms, the timer will overflow, causing the 8096 to be reset—resetting reinitializes the 8096. This feature makes it virtually impossible for the 8096 to become lost in a program for too long. For development purposes, the reset terminal can be connected to  $V_{CC}$  to disable the watchdog timer.

### More I/O—and faster

Correlating events in real time is one of the most important considerations in computer-based control system design. Another common requirement is generating pulses and pulse trains to drive actuators. Most single-chip microcontrollers support such operations by having one or more timer/event counters under software control. The 8096, on the other hand, offers a complete integrated subsystem to perform these functions. Called the high-speed I/O unit, it is intended to be an integrated subsystem, but it can be viewed as separate units for input and output.

Figure 3 shows the block diagram of the high-speed input unit. Its major components are a 16-bit timer, a programmable change detector and a first-in, first-out (FIFO) memory. Also included are several registers used by the software to control the high-speed input unit.

The read-only timer is cleared by the system reset and incremented once every eight CPU cycles (every  $1.6 \mu s$  with a 15-MHz crystal). When the timer overflows—rolls over from  $FFFF_H$  to  $0000_H$ —a status bit is set and an interrupt is generated. The change detector monitors four pins on the 8096 and looks for predefined changes. Change definitions are controlled by the high-speed input unit's mode register, which is set by the software. This register contains a 2-bit field for each of the four high-speed inputs. Using the fields, a programmer can select the type of change for each input. Fields are encoded in one of four ways:

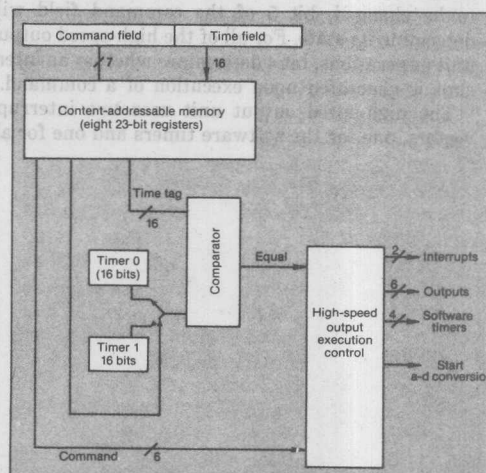
- 00 defines positive transitions divided by 8.
- 01 defines positive transitions.
- 10 defines negative transitions.
- 11 defines positive and negative transitions.

Each high-speed input can be disabled through a second control register. When this is done, inputs of the high-speed input unit become available as digital input pins or, if required, two of the pins can be connected to the high-speed output unit.

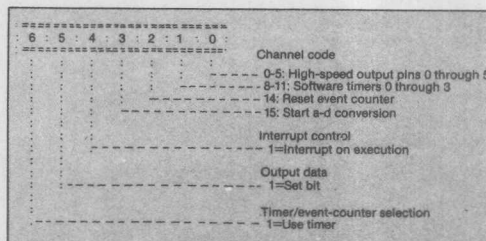
As the block diagram in Fig. 4 shows, the high-speed output unit uses the same timer as the input

unit and also has a 16-bit event counter. The read-only event counter is similar to the timer in that it can be read at any time, generates an overflow-interrupt or status indication, and cannot be written into. It differs from the timer, in that its reset and clock sources, instead of being fixed by hardware, can be selected under software control.

Two of the 8096's pins are dedicated to the event counter. A positive-going pulse on ECRST (Event Counter Reset) clears the counter, and either edge of a pulse applied to ECCLK (Event Counter Clock) increments the counter. A programmer has the option of using  $HSI_0$  instead of ECRST or  $HSI_1$  instead of ECCLK. These options are available by setting the appropriate bits in the I/O control register. The event counter can also be cleared under software control either directly, by setting a bit in the I/O control register, or indirectly, using the high-



4. The other half of the high-speed I/O subsystem is the output unit. Using a content-addressable memory to store so-called time-field data, the unit's logic matches this information with timer or event-counter operations.



5. In the content-addressable memory of the high-speed output unit, 23-bit words are broken down into a 16-bit time field and a 7-bit command field. Command-field encoding defines the output unit's operating mode.



The FIFO register of the high-speed input unit is replaced by a content-addressable memory in the output unit. The memory contains a file of eight 23-bit registers. The 23 bits are divided into a 16-bit time field and a 7-bit command field. Control logic continually scans each location in the memory to determine whether its time field matches either the timer or the event counter as selected by one of the seven bits in the command field. When a match is found, the remaining six bits in the command field are executed.

#### What the bits do

The encoding of the command field bits and their functions are shown in Fig. 5. Four-bit channel code selects the output unit's operation. For example, the event counter can be reset or an a-d conversion can be initiated. If one of the high-speed output pins is to be changed, bit 5 of the command field will determine its state. For all of the high-speed output unit's operations, bit 4 determines whether an interrupt is generated upon execution of a command.

The high-speed output unit uses two interrupt vectors, one for the software timers and one for all

an I/O status register to determine which of the four timers caused the interrupt. The ability of the command field to trigger an a-d conversion allows measurements to be made at precise moments, an absolute necessity in digital signal processing. Also, the ability to reset a count when it reaches a preset limit allows the simple implementation of a modulo-N counter. This is useful, for example, in a crankshaft position-sensor application that generates 214 pulses per revolution.

The eight locations in the content-addressable memory's file are scanned at the rate of one CPU cycle per location. At a 15-MHz clock rate, all eight locations will be scanned within 1.6  $\mu$ s. A high-speed output unit's command is executed as soon as it finds a time match. As each command is executed, it is removed from the content-addressable memory to make room for a new command, which is sent in from the input holding register.

Because of the extensive functions built into the 8096, a standard 40-lead DIP is far too small; the 8096 is housed in a 68-pin JEDEC package. Alternatively, it is supplied in a 48-pin DIP. □

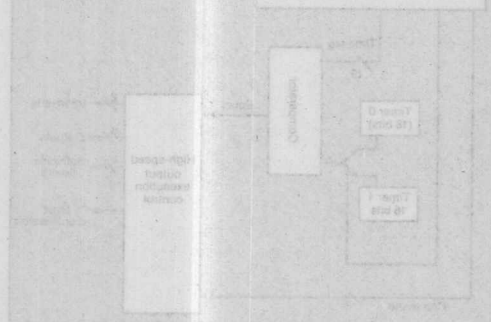


Figure 3 shows the block diagram of the high-speed input unit. Its major components are a 16-bit timer, a programmable channel detector and a first-in, first-out (FIFO) memory. Also included are several registers used by the software to control the high-speed input unit.

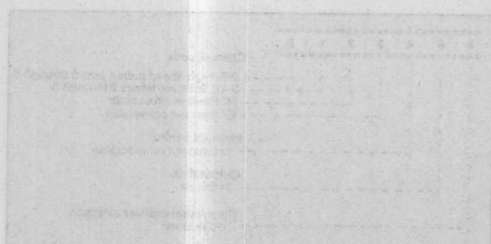


Figure 4 shows the block diagram of the high-speed output unit. Using a content-addressable memory to store so-called time-field data, the unit's logic selects this information with timer or event-counter operations.

The read-only timer is cleared by the system reset and incremented once every eight CPU cycles (every 1.5  $\mu$ s with a 15-MHz crystal). When the timer overflows—rolls over from FFFF<sub>16</sub> to 0000<sub>16</sub>—a status bit is set and an interrupt is generated. The change detector monitors four pins on the 8096 and looks for predefined changes. Change definitions are controlled by the high-speed input unit's mode register, which is set by the software. This register contains a 2-bit field for each of the four high-speed inputs. Using the fields, a programmer can select the type of change for each input. Fields are encoded in one of four ways:

- 00 defines positive transitions divided by 8.
  - 01 defines positive transitions.
  - 10 defines negative transitions.
  - 11 defines positive and negative transitions.
- Each high-speed input can be disabled through a second control register. When this is done, inputs of the high-speed input unit become available as digital input pins or, if required, two of the pins can be connected to the high-speed output unit.

As the block diagram in Fig. 4 shows, the high-speed output unit uses the same timer as the input unit. The high-speed output unit's command is executed as soon as it finds a time match. As each command is executed, it is removed from the content-addressable memory to make room for a new command, which is sent in from the input holding register.

The Single Component  
**MCS®-48 System**

---

**12**

12

The Single Component  
MCS®-48 System

# CHAPTER 12

## THE SINGLE COMPONENT MCS<sup>®</sup>-48 SYSTEM

### 12.0 INTRODUCTION

Sections 12.1 through 12.4 describe in detail the functional characteristics of the 8748H and 8749H EPROM, 8048AH/8049AH/8050AH ROM, and 8035AHL/8039AHL/8040-AHL CPU only single component microcomputers. Unless otherwise noted, details within these sections apply to all versions. This chapter is limited to those functions useful in single-chip implementations of the MCS<sup>®</sup>-48. Chapter 14 discusses functions which allow expansion of program memory, data memory, and input output capability.

### 12.1 ARCHITECTURE

The following sections break the MCS-48 Family into functional blocks and describe each in detail. The following description will use the 8048AH as the representative product for the family. See Figure 14.1.

#### 12.1.1 Arithmetic Section

The arithmetic section of the processor contains the basic data manipulation functions of the 8048AH and can be divided into the following blocks:

- Arithmetic Logic Unit (ALU)
- Accumulator
- Carry Flag
- Instruction Decoder

In a typical operation data stored in the accumulator is combined in the ALU with data from another source on the internal bus (such as a register or I/O port) and the result is stored in the accumulator or another register.

The following is more detailed description of the function of each block.

#### INSTRUCTION DECODER

The operation code (op code) portion of each program instruction is stored in the Instruction Decoder and converted to outputs which control the function of each of the blocks of the Arithmetic Section. These lines control the source of data and the destination register as well as the function performed in the ALU.

#### ARITHMETIC LOGIC UNIT

The ALU accepts 8-bit data words from one or two sources and generates an 8-bit result under control of the Instruction Decoder. The ALU can perform the following functions:

- Add With or Without Carry
- AND, OR, Exclusive OR
- Increment/Decrement
- Bit Complement
- Rotate Left, Right
- Swap Nibbles
- BCD Decimal Adjust

If the operation performed by the ALU results in a value represented by more than 8 bits (overflow of most significant bit), a Carry Flag is set in the Program Status Word.

#### ACCUMULATOR

The accumulator is the single most important data register in the processor, being one of the sources of input to the ALU and often the destination of the result of operations performed in the ALU. Data to and from I/O ports and memory also normally passes through the accumulator.

#### 12.1.2 Program Memory

Resident program memory consists of 1024, 2048, or 4096 words eight bits wide which are addressed by the program counter. In the 8748H and the 8749H this memory is user programmable and erasable EPROM; in the 8048AH/8049AH/8050AH the memory is ROM which is mask programmable at the factory. The 8035AHL/8039AHL/8040AHL has no internal program memory and is used with external memory devices. Program code is completely interchangeable among the various versions. To access the upper 2K of program memory in the 8050AH, and other MCS-48 devices, a select memory bank and a JUMP or CALL instruction must be executed to cross the 2K boundary.

There are three locations in Program Memory of special importance as shown in Figure 12.2.

##### LOCATION 0

Activating the Reset line of the processor causes the first instruction to be fetched from location 0.

##### LOCATION 3

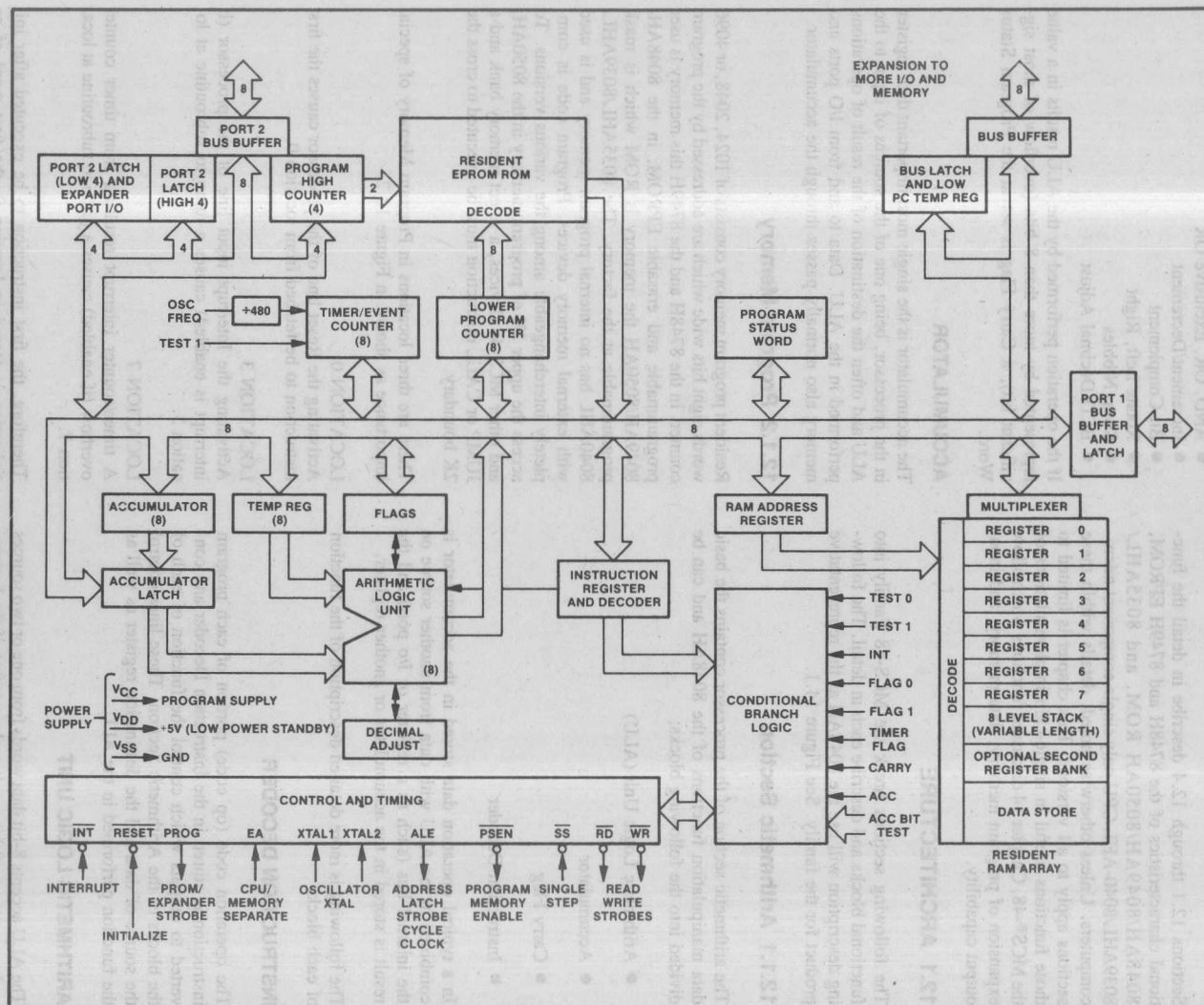
Activating the Interrupt input line of the processor (if interrupt is enabled) causes a jump to subroutine at location 3.

##### LOCATION 7

A timer/counter interrupt resulting from timer counter overflow (if enabled) causes a jump to subroutine at location 7.

Therefore, the first instruction to be executed after initialization is stored in location 0, the first word of an external interrupt service subroutine is stored in location 3, and the first word of a timer/counter service routines





is stored in location 7. Program memory can be used to store constants as well as program instructions. Instructions such as MOVP and MOVP3 allow easy access to data "lookup" tables.

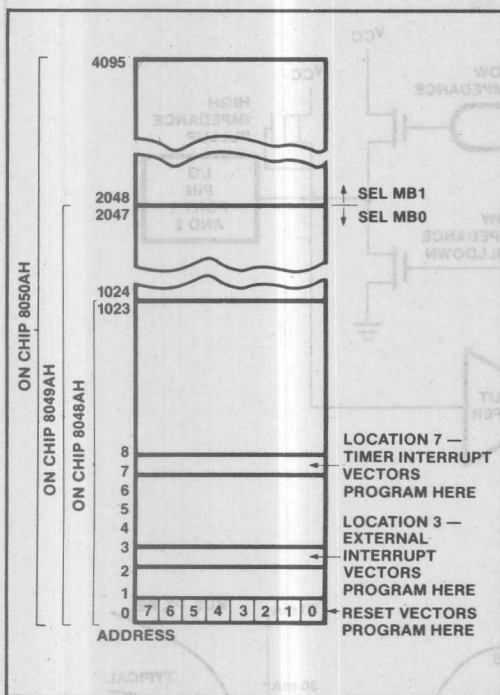


Figure 12-2. Program Memory Map

### 12.1.3 Data Memory

Resident data memory is organized as 64, 128, or 256 by 8-bits wide in the 8048AH, 8049AH and 8050AH. All locations are indirectly addressable through either of two RAM Pointer Registers which reside at address 0 and 1 of the register array. In addition, as shown in Figure 12-3, the first 8 locations (0-7) of the array are designated as working registers and are directly addressable by several instructions. Since these registers are more easily addressed, they are usually used to store frequently accessed intermediate results. The DJNZ instruction makes very efficient use of the working registers as program loop counters by allowing the programmer to decrement and test the register in a single instruction.

By executing a Register Bank Switch instruction (SEL RB) RAM locations 24-31 are designated as the working

registers in place of locations 0-7 and are then directly addressable. This second bank of working registers may be used as an extension of the first bank or reserved for use during interrupt service subroutines allowing the registers of Bank 0 used in the main program to be instantly "saved" by a Bank Switch. Note that if this second bank is not used, locations 24-31 are still addressable as general purpose RAM. Since the two RAM pointer Registers R0 and R1 are a part of the working register array, bank switching effectively creates two more pointer registers (R0' and R1') which can be used with R0 and R1 to easily access up to four separate working areas in RAM at one time. RAM locations (8-23) also serve a dual role in that they contain the program counter stack as explained in Section 12.1.6. These locations are addressed by the Stack Pointer during subroutine calls as well as by RAM Pointer Registers R0 and R1. If the level of subroutine nesting is less than 8, all stack registers are not required and can be used as general purpose RAM locations. Each level of subroutine nesting not used provides the user with two additional RAM locations.

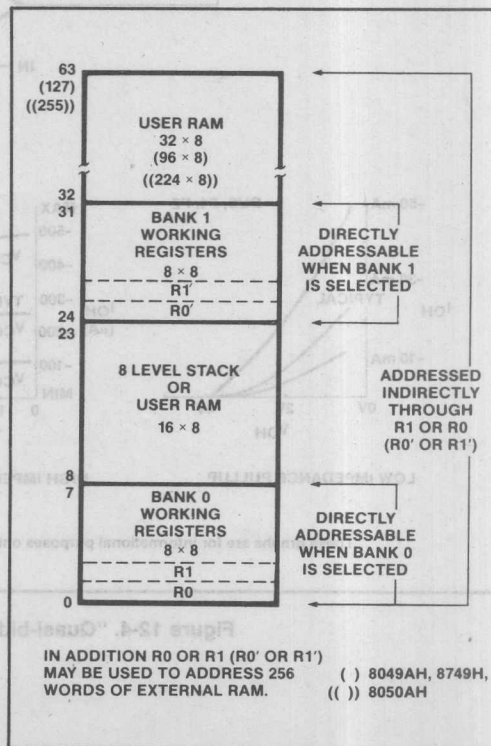


Figure 12-3. Data Memory Map

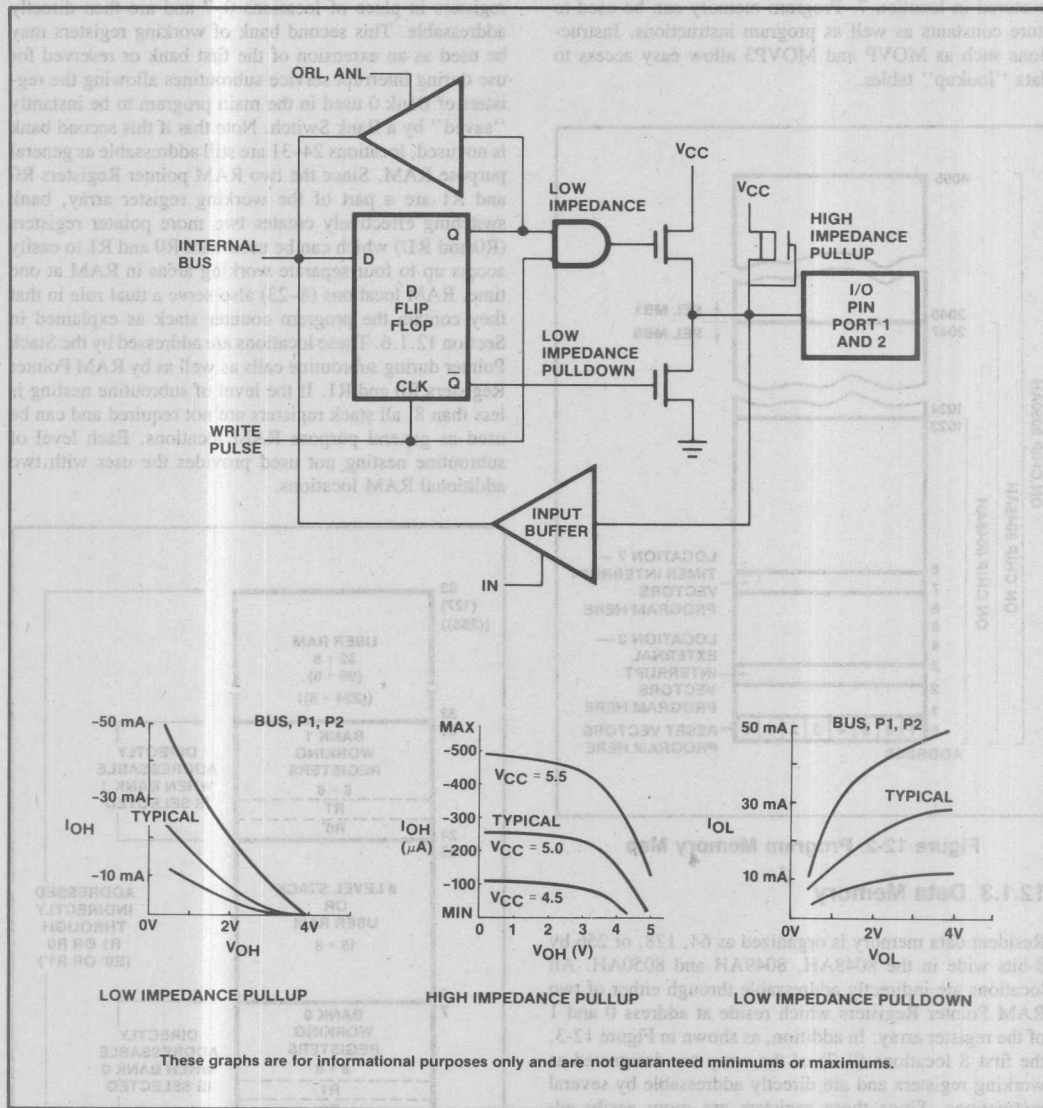


Figure 12-4. "Quasi-bidirectional" Port Structure

### 12.1.4 Input/Output

The 8048AH has 27 lines which can be used for input or output functions. These lines are grouped as 3 ports of 8 lines each which serve as either inputs, outputs or bidirectional ports and 3 "test" inputs which can alter program sequences when tested by conditional jump instructions.

### PORTS 1 AND 2

Ports 1 and 2 are each 8 bits wide and have identical characteristics. Data written to these ports is statically latched and remains unchanged until rewritten. As input ports these lines are non-latching, i.e., inputs must be present until read by an input instruction. Inputs are fully TTL compatible and outputs will drive one standard TTL load.

The lines of ports 1 and 2 are called quasi-bidirectional because of a special output circuit structure which allows each line to serve as an input, and output, or both even though outputs are statically latched. Figure 12-4 shows the circuit configuration in detail. Each line is continuously pulled up to  $V_{CC}$  through a resistive device of relatively high impedance.

This pullup is sufficient to provide the source current for a TTL high level yet can be pulled low by a standard TTL gate thus allowing the same pin to be used for both input and output. To provide fast switching times in a "0" to "1" transition a relatively low impedance device is switched in momentarily ( $\approx 1/5$  of a machine cycle) whenever a "1" is written to the line. When a "0" is written to the line a low impedance device overcomes the light pullup and provides TTL current sinking capability. Since the pulldown transistor is a low impedance device a "1" must first be written to any line which is to be used as an input. Reset initializes all lines to the high impedance "1" state.

It is important to note that the ORL and the ANL are read/write operations. When executed, the  $\mu C$  "reads" the port, modifies the data according to the instruction, then "writes" the data back to the port. The "writing" (essentially an OUTL instruction) enables the low impedance pull-up momentarily again even if the data was unchanged from a "1." This specifically applies to configurations that have inputs and outputs mixed together on the same port. See also section 13.7.

### BUS

Bus is also an 8-bit port which is a true bidirectional port with associated input and output strobes. If the bidirectional feature is not needed, Bus can serve as either a

statically latched output port or non-latching input port. Input and output lines on this port cannot be mixed however.

As a static port, data is written and latched using the OUTL instruction and inputted using the INS instruction. The INS and OUTL instructions generate pulses on the corresponding  $\overline{RD}$  and  $\overline{WR}$  output strobe lines; however, in the static port mode they are generally not used. As a bidirectional port the MOVX instructions are used to read and write the port. A write to the port generates a pulse on the  $\overline{WR}$  output line and output data is valid at the trailing edge of  $\overline{WR}$ . A read of the port generates a pulse on the  $\overline{RD}$  output line and input data must be valid at the trailing edge of  $\overline{RD}$ . When not being written or read, the BUS lines are in a high impedance state. See also sections 13.6 and 13.7.

### 12.1.5 Test and INT Inputs

Three pins serve as inputs and are testable with the conditional jump instruction. These are T0, T1, and  $\overline{INT}$ . These pins allow inputs to cause program branches without the necessity to load an input port into the accumulator. The T0, T1, and  $\overline{INT}$  pins have other possible functions as well. See the pin description in Section 12.2.

### 12.1.6 Program Counter and Stack

The Program Counter is an independent counter while the Program Counter Stack is implemented using pairs of registers in the Data Memory Array. Only 10, 11, or 12 bits of the Program Counter are used to address the 1024, 2048, or 4096 words of on-board program memory of the 8048AH, 8049AH, or 8050AH, while the most significant bits can be used for external Program Memory fetches. See Figure 12.5. The Program Counter is initialized to zero by activating the Reset line.

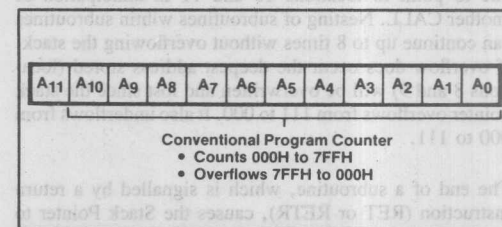


Figure 12-5. Program Counter

An interrupt or CALL to a subroutine causes the contents of the program counter to be stored in one of the 8 register pairs of the Program Counter Stack as shown in Figure 12-6. The pair to be used is determined by a 3-bit Stack Pointer which is part of the Program Status Word (PSW).



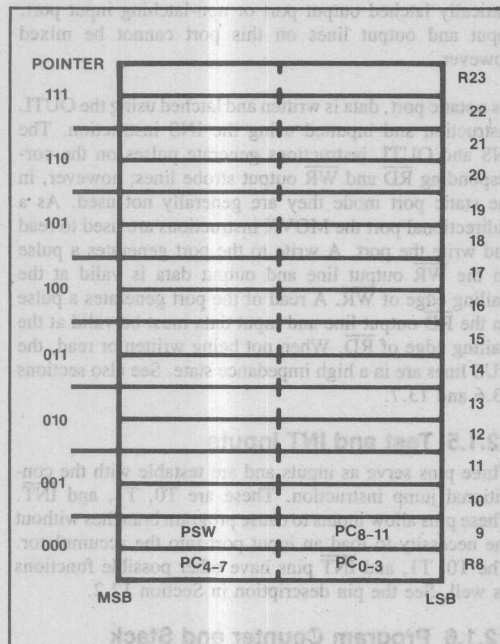


Figure 12-6. Program Counter Stack

Data RAM locations 8-23 are available as stack registers and are used to store the Program Counter and 4 bits of PSW as shown in Figure 12-6. The Stack Pointer when initialized to 000 points to RAM locations 8 and 9. The first subroutine jump or interrupt results in the program counter contents being transferred to locations 8 and 9 of the RAM array. The stack pointer is then incremented by one to point to locations 10 and 11 in anticipation of another CALL. Nesting of subroutines within subroutines can continue up to 8 times without overflowing the stack. If overflow does occur the deepest address stored (locations 8 and 9) will be overwritten and lost since the stack pointer overflows from 111 to 000. It also underflows from 000 to 111.

The end of a subroutine, which is signalled by a return instruction (RET or RETR), causes the Stack Pointer to be decremented and the contents of the resulting register pair to be transferred to the Program Counter.

### 12.1.7 Program Status Word

An 8-bit status word which can be loaded to and from the accumulator exists called the Program Status Word (PSW). Figure 12-7 shows the information available in

the word. The Program Status Word is actually a collection of flip-flops throughout the machine which can be read or written as a whole. The ability to write to PSW allows for easy restoration of machine status after a power down sequence.

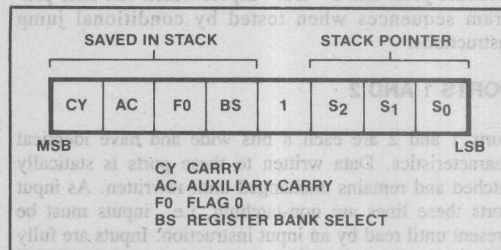


Figure 12-7. Program Status Word (PSW)

The upper four bits of PSW are stored in the Program Counter Stack with every call to subroutine or interrupt vector and are optionally restored upon return with the RETR instruction. The RET return instruction does not update PSW.

The PSW bit definitions are as follows:

- Bits 0-2: Stack Pointer bits ( $S_0$ ,  $S_1$ ,  $S_2$ )
- Bit 3: Not used ('1' level when read)
- Bit 4: Working Register Bank Switch Bit (BS)  
0 = Bank 0  
1 = Bank 1
- Bit 5: Flag 0 bit (F0) user controlled flag which can be complemented or cleared, and tested with the conditional jump instruction JF0.
- Bit 6: Auxiliary Carry (AC) carry bit generated by an ADD instruction and used by the decimal adjust instruction DA A.
- Bit 7: Carry (CY) carry flag which indicates that the previous operation has resulted in overflow of the accumulator.

### 12.1.8 Conditional Branch Logic

The conditional branch logic within the processor enables several conditions internal and external to the processor to be tested by the users program. By using the conditional jump instruction the conditions that are listed in Table 12-1 can effect a change in the sequence of the program execution.

Table 12-1

Device Testable	Jump Conditions (Jump On)	
	All zeros	not all zeros
Accumulator	—	1
Accumulator Bit	—	1
Carry Flag	0	1
User Flags (F0, F1)	—	1
Timer Overflow Flag	—	1
Test Inputs (T0, T1)	0	1
Interrupt Input (INT)	0	—

### 12.1.9 Interrupt

An interrupt sequence is initiated by applying a low "0" level input to the INT pin. Interrupt is level triggered and active low to allow "WIRE ORing" of several interrupt sources at the input pin. Figure 12-8 shows the interrupt logic of the 8048AH. The Interrupt line is sampled every instruction cycle and when detected causes a "call to subroutine" at location 3 in program memory as soon as all cycles of the current instruction are complete. On 2-cycle instructions the interrupt line is sampled on the 2nd cycle only.  $\overline{\text{INT}}$  must be held low for at least 3 machine cycles to ensure proper interrupt operations. As in any CALL to subroutine, the Program Counter and Program Status word are saved in the stack. For a description of this operation see the previous section, Program Counter and Stack. Program Memory location 3 usually contains an unconditional jump to an interrupt service subroutine elsewhere in program memory. The end of an interrupt service subroutine is signalled by the execution of a Return and Restore Status instruction RETR. The interrupt system is single level in that once an interrupt is detected all further interrupt requests are ignored until execution of an RETR reenables the interrupt input logic. This occurs at the beginning of the second cycle of the RETR instruction. This sequence holds true also for an internal interrupt generated by timer overflow. If an internal timer/counter generated interrupt and an external interrupt are detected at the same time, the external source will be recognized. See the following Timer/Counter section for a description of timer interrupt. If needed, a second external interrupt can be created by enabling the timer/counter interrupt, loading FFH in the Counter (ones less than terminal count), and enabling the event counter mode. A "1" to "0" transition on the T1 input will then cause an interrupt vector to location 7.

### INTERRUPT TIMING

The interrupt input may be enabled or disabled under Program Control using the EN I and DIS I instructions. Interrupts are disabled by Reset and remain so until en-

abled by the users program. An interrupt request must be removed before the RETR instruction is executed upon return from the service routine otherwise the processor will re-enter the service routine immediately. Many peripheral devices prevent this situation by resetting their interrupt request line whenever the processor accesses (Reads or Writes) the peripherals data buffer register. If the interrupting device does not require access by the processor, one output line of the 8048AH may be designated as an "interrupt acknowledge" which is activated by the service subroutine to reset the interrupt request. The INT pin may also be tested using the conditional jump instruction JNI. This instruction may be used to detect the presence of a pending interrupt before interrupts are enabled. If interrupt is left disabled,  $\overline{\text{INT}}$  may be used as another test input like T0 and T1.

### 12.1.10 Time/Counter

The 8048AH contains a counter to aid the user in counting external events and generating accurate time delays without placing a burden on the processor for these functions. In both modes the counter operation is the same, the only difference being the source of the input to the counter. The timer/event counter is shown in Figure 12-9.

### COUNTER

The 8-bit binary counter is presettable and readable with two MOV instructions which transfer the contents of the accumulator to the counter and vice versa. The counter content may be affected by Reset and should be initialized by software. The counter is stopped by a Reset or STOP TCNT instruction and remains stopped until started as a timer by a START T instruction or as an event counter by a START CNT instruction. Once started the counter will increment to this maximum count (FF) and overflow to zero continuing its count until stopped by a STOP TCNT instruction or Reset.

The increment from maximum count to zero (overflow) results in the setting of an overflow flag flip-flop and in the generation of an interrupt request. The state of the overflow flag is testable with the conditional jump instruction JTF. The flag is reset by executing a JTF or by Reset. The interrupt request is stored in a latch and then ORed with the external interrupt input INT. The timer interrupt may be enabled or disabled independently of external interrupt by the EN TCNT1 and DIS TCNT1 instructions. If enabled, the counter overflow will cause a subroutine call to location 7 where the timer or counter service routine may be stored.

If timer and external interrupts occur simultaneously, the external source will be recognized and the Call will be to

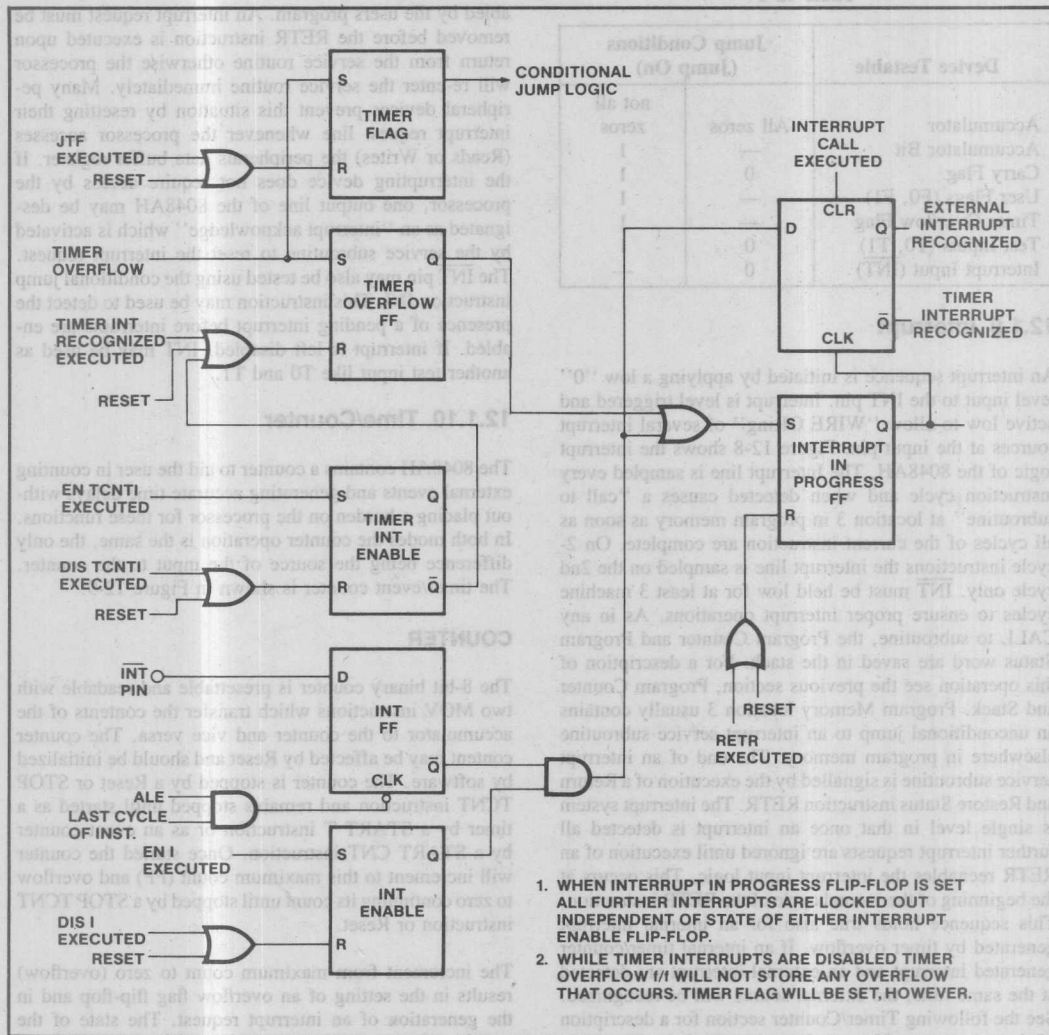


Figure 12-8. Interrupt Logic

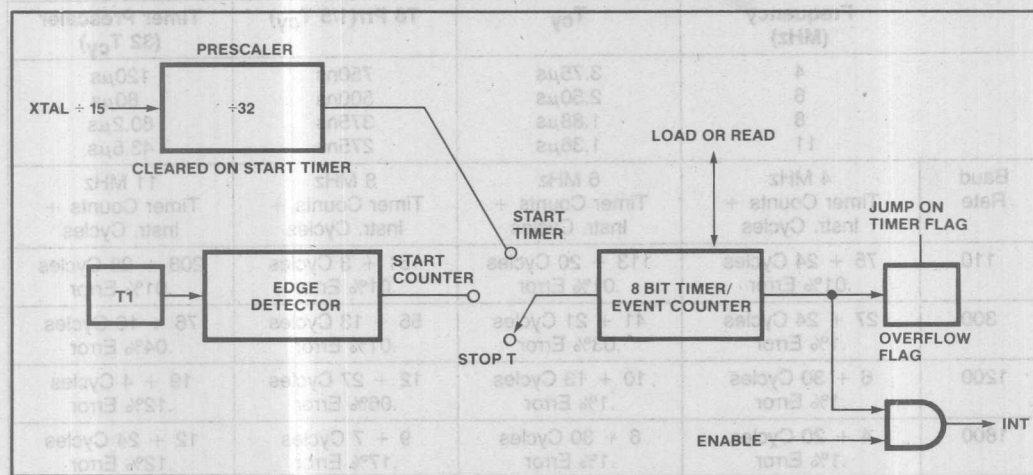


Figure 12-9. Timer/Event Counter

location 3. Since the timer interrupt is latched it will remain pending until the external device is serviced and immediately be recognized upon return from the service routine. The pending timer interrupt is reset by the Call to location 7 or may be removed by executing a DIS TCNT1 instruction.

#### AS AN EVENT COUNTER

Execution of a START CNT instruction connects the T1 input pin to the counter input and enables the counter. The T1 input is sampled at the beginning of state 3 or in later MCS-48 devices in state time 4. Subsequent high to low transitions on T1 will cause the counter to increment. T1 must be held low for at least 1 machine cycle to insure it won't be missed. The maximum rate at which the counter may be incremented is once per three instruction cycles (every 5.7  $\mu$ sec when using an 8 MHz crystal) — there is no minimum frequency. T1 input must remain high for at least 1/5 machine cycle after each transition.

#### AS A TIMER

Execution of a START T instruction connects an internal clock to the counter input and enables the counter. The internal clock is derived bypassing the basic machine cycle clock through a  $\div 32$  prescaler. The prescaler is reset during the START T instruction. The resulting clock increments the counter every 32 machine cycles. Various delays from 1 to 256 counts can be obtained by presetting the counter and detecting overflow. Times longer than 256 counts may be achieved by accumulating multiple overflows in a register under software control. For time res-

olution less than 1 count an external clock can be applied to the T1 input and the counter operated in the event counter mode. ALE divided by 3 or more can serve as this external clock. Very small delays or "fine tuning" of larger delays can be easily accomplished by software delay loops.

Often a serial link is desirable in an MCS-48 family member. Table 12-2 lists the timer counts and cycles needed for a specific baud rate given a crystal frequency.

#### 12.1.11 Clock and Timing Circuits

Timing generation for the 8048AH is completely self-contained with the exception of a frequency reference which can be XTAL, ceramic resonator, or external clock source. The Clock and Timing circuitry can be divided into the following functional blocks.

##### OSCILLATOR

The on-board oscillator is a high gain parallel resonant circuit with a frequency range of 1 to 11 MHz. The X1 external pin is the input to the amplifier stage while X2 is the output. A crystal or ceramic resonator connected between X1 and X2 provides the feedback and phase shift required for oscillation. If an accurate frequency reference is not required, ceramic resonator may be used in place of the crystal.

For accurate clocking, a crystal should be used. An externally generated clock may also be applied to X1-X2 as the frequency source. See the data sheet for more information.



Table 12-2. Baud Rate Generation

	Frequency (MHz)	T <sub>cy</sub>	T0 Prr(1/5 T <sub>cy</sub> )	Timer Prescaler (32 T <sub>cy</sub> )
	4	3.75μs	750ns	120μs
	6	2.50μs	500ns	80μs
	8	1.88μs	375ns	60.2μs
	11	1.36μs	275ns	43.5μs
Baud Rate	4 MHz Timer Counts + Instr. Cycles	6 MHz Timer Counts + Instr. Cycles	8 MHz Timer Counts + Instr. Cycles	11 MHz Timer Counts + Instr. Cycles
110	75 + 24 Cycles .01% Error	113 + 20 Cycles .01% Error	151 + 3 Cycles .01% Error	208 + 28 Cycles .01% Error
300	27 + 24 Cycles .1% Error	41 + 21 Cycles .03% Error	55 + 13 Cycles .01% Error	76 + 18 Cycles .04% Error
1200	6 + 30 Cycles .1% Error	10 + 13 Cycles .1% Error	12 + 27 Cycles .06% Error	19 + 4 Cycles .12% Error
1800	4 + 20 Cycles .1% Error	6 + 30 Cycles .1% Error	9 + 7 Cycles .17% Error	12 + 24 Cycles .12% Error
2400	3 + 15 Cycles .1% Error	5 + 6 Cycles .4% Error	6 + 24 Cycles .29% Error	9 + 18 Cycles .12% Error
4800	1 + 23 Cycles 1.0% Error	2 + 19 Cycles .4% Error	3 + 14 Cycles .74% Error	4 + 25 Cycles .12% Error

## STATE COUNTER

The output of the oscillator is divided by 3 in the State Counter to create a clock which defines the state times of the machine (CLK). CLK can be made available on the external pin T0 by executing an ENTO CLK instruction. The output of CLK on T0 is disabled by Reset of the processor.

## CYCLE COUNTER

CLK is then divided by 5 in the Cycle Counter to provide a clock which defines a machine cycle consisting of 5 machine states as shown in Figure 12-10. Figure 12-11 shows the different internal operations as divided into the machine states. This clock is called Address Latch Enable (ALE) because of its function in MCS-48 systems with external memory. It is provided continuously on the ALE output pin.

### 12.1.12 Reset

The reset input provides a means for initialization for the processor. This Schmitt-trigger input has an internal pull-up device which in combination with an external 1 μf capacitor provides an internal reset pulse of sufficient length to guarantee all circuitry is reset, as shown in Figure 12-12. If the reset pulse is generated externally the RESET pin must be held low for at least 10 milliseconds after the

power supply is within tolerance. Only 5 machine cycles (6.8 μs @ 11 MHz) are required if power is already on and the oscillator has stabilized. ALE and PSEN (if EA = 1) are active while in Reset.

Reset performs the following functions:

- 1) Sets program counter to zero.
- 2) Sets stack pointer to zero.
- 3) Selects register bank 0.
- 4) Selects memory bank 0.
- 5) Sets BUS to high impedance state (except when EA = 5V).
- 6) Sets Ports 1 and 2 to input mode.
- 7) Disables interrupts (timer and external).
- 8) Stops timer.
- 9) Clears timer flag.
- 10) Clears F0 and F1.
- 11) Disables clock output from T0.

## SINGLE COMPONENT MCS®-48 SYSTEM

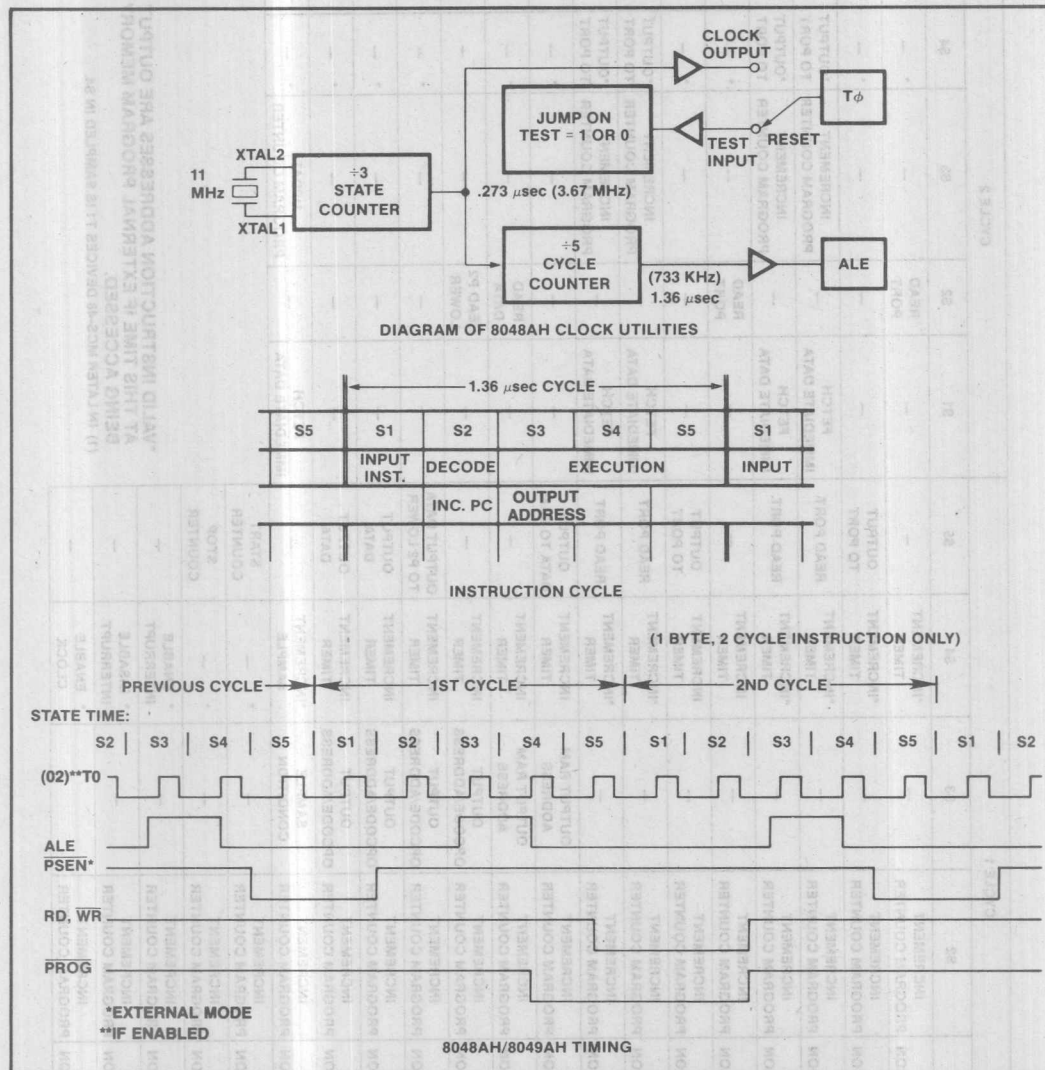


Figure 12-10. MCS®-48 Timing Generation and Cycle Timing

### 12.1.13 Single-Step

This feature, as pictured in Figure 12-13, provides the user with a debug capability in that the processor can be stepped through the program one instruction at a time. While stopped, the address of the next instruction to be fetched is available concurrently on BUS and the lower

half of Port 2. The user can therefore follow the program through each of the instruction steps. A timing diagram, showing the interaction between output ALE and input SS, is shown. The BUS buffer contents are lost during single step; however, a latch may be added to reestablish the lost I/O capability if needed. Data is valid at the leading edge of ALE.

INSTRUCTION	CYCLE 1					CYCLE 2				
	S1	S2	S3	S4	S5	S1	S2	S3	S4	S5
IN A,P	FETCH INSTRUCTION	INCREMENT PROGRAM COUNTER	—	*INCREMENT TIMER	—	—	READ PORT	—	* —	—
OUTL P,A	FETCH INSTRUCTION	INCREMENT PROGRAM COUNTER	—	*INCREMENT TIMER	OUTPUT TO PORT	—	—	—	* —	—
ANL P, = DATA	FETCH INSTRUCTION	INCREMENT PROGRAM COUNTER	—	*INCREMENT TIMER	READ PORT	FETCH IMMEDIATE DATA	—	INCREMENT PROGRAM COUNTER	*OUTPUT TO PORT	—
ORL P, = DATA	FETCH INSTRUCTION	INCREMENT PROGRAM COUNTER	—	*INCREMENT TIMER	READ PORT	FETCH IMMEDIATE DATA	—	INCREMENT PROGRAM COUNTER	*OUTPUT TO PORT	—
INS A, BUS	FETCH INSTRUCTION	INCREMENT PROGRAM COUNTER	—	INCREMENT TIMER	—	—	READ PORT	—	* —	—
OUTL BUS, A	FETCH INSTRUCTION	INCREMENT PROGRAM COUNTER	—	INCREMENT TIMER	OUTPUT TO PORT	—	—	—	* —	—
ANL BUS, = DATA	FETCH INSTRUCTION	INCREMENT PROGRAM COUNTER	—	*INCREMENT TIMER	READ PORT	FETCH IMMEDIATE DATA	—	INCREMENT PROGRAM COUNTER	*OUTPUT TO PORT	—
ORL BUS, = DATA	FETCH INSTRUCTION	INCREMENT PROGRAM COUNTER	—	*INCREMENT TIMER	READ PORT	FETCH IMMEDIATE DATA	—	INCREMENT PROGRAM COUNTER	*OUTPUT TO PORT	—
MOVX @ R,A	FETCH INSTRUCTION	INCREMENT PROGRAM COUNTER	OUTPUT RAM ADDRESS	INCREMENT TIMER	OUTPUT DATA TO RAM	—	—	—	* —	—
MOVX A,@R	FETCH INSTRUCTION	INCREMENT PROGRAM COUNTER	OUTPUT RAM ADDRESS	INCREMENT TIMER	—	—	READ DATA	—	* —	—
MOVD A,P <sub>1</sub>	FETCH INSTRUCTION	INCREMENT PROGRAM COUNTER	OUTPUT OPCODE/ADDRESS	INCREMENT TIMER	—	—	READ P2 LOWER	—	* —	—
MOVD P <sub>1</sub> ,A	FETCH INSTRUCTION	INCREMENT PROGRAM COUNTER	OUTPUT OPCODE/ADDRESS	INCREMENT TIMER	OUTPUT DATA TO P2 LOWER	—	—	—	* —	—
ANLD P,A	FETCH INSTRUCTION	INCREMENT PROGRAM COUNTER	OUTPUT OPCODE/ADDRESS	INCREMENT TIMER	OUTPUT DATA	—	—	—	* —	—
ORLD P,A	FETCH INSTRUCTION	INCREMENT PROGRAM COUNTER	OUTPUT OPCODE/ADDRESS	INCREMENT TIMER	OUTPUT DATA	—	—	—	* —	—
J(CONDITIONAL)	FETCH INSTRUCTION	INCREMENT PROGRAM COUNTER	SAMPLE CONDITION	*INCREMENT SAMPLE	—	FETCH IMMEDIATE DATA	—	UPDATE PROGRAM COUNTER	* —	—
STRT T	FETCH INSTRUCTION	INCREMENT PROGRAM COUNTER	—	* —	START COUNTER	<p>*VALID INSTRUCTION ADDRESSES ARE OUTPUT AT THIS TIME IF EXTERNAL PROGRAM MEMORY IS BEING ACCESSED.</p> <p>(1) IN LATER MCS-48 DEVICES T1 IS SAMPLED IN S4.</p>				
STOP TCNT	FETCH INSTRUCTION	INCREMENT PROGRAM COUNTER	—	* —	STOP COUNTER					
ENI	FETCH INSTRUCTION	INCREMENT PROGRAM COUNTER	—	* ENABLE INTERRUPT	—					
DIS I	FETCH INSTRUCTION	INCREMENT PROGRAM COUNTER	—	* DISABLE INTERRUPT	—					
ENTO CLK	FETCH INSTRUCTION	INCREMENT PROGRAM COUNTER	—	* ENABLE CLOCK	—					

Figure 12-11. 8048AH/8049AH Instruction Timing Diagram

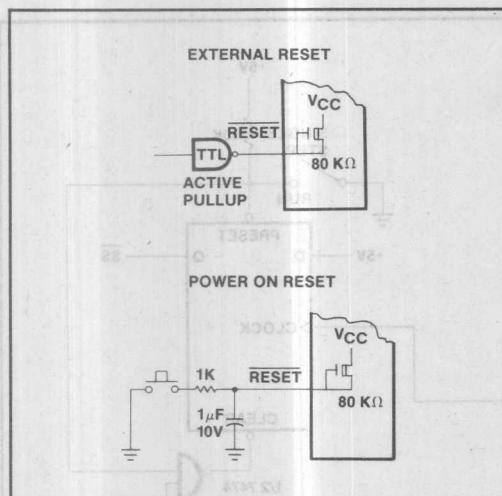


Figure 12-12

## TIMING

The 8048AH operates in a single-step mode as follows:

- 1) The processor is requested to stop by applying a low level on  $\overline{SS}$ .
- 2) The processor responds by stopping during the address fetch portion of the next instruction. If a double cycle instruction is in progress when the single step command is received, both cycles will be completed before stopping.
- 3) The processor acknowledges it has entered the stopped state by raising ALE high. In this state (which can be maintained indefinitely) the address of the next instruction to be fetched is present on BUS and the lower half of port 2.
- 4)  $\overline{SS}$  is then raised high to bring the processor out of the stopped mode allowing it to fetch the next instruction. The exit from stop is indicated by the processor bringing ALE low.
- 5) To stop the processor at the next instruction  $\overline{SS}$  must be brought low again soon after ALE goes low. If  $\overline{SS}$  is left high the processor remains in a "Run" mode.

A diagram for implementing the single-step function of the 8748H is shown in Figure 12-13. D-type flip-flop with preset and clear is used to generate  $\overline{SS}$ . In the run mode  $\overline{SS}$  is held high by keeping the flip-flop preset (preset has precedence over the clear input). To enter single step, preset is removed allowing ALE to bring  $\overline{SS}$  low via the

clear input. ALE should be buffered since the clear input of an SN7474 is the equivalent of 3 TTL loads. The processor is now in the stopped state. The next instruction is initiated by clocking a "1" into the flip-flop. This "1" will not appear on  $\overline{SS}$  unless ALE is high removing clear from the flip-flop. In response to  $\overline{SS}$  going high the processor begins an instruction fetch which brings ALE low resetting  $\overline{SS}$  through the clear input and causing the processor to again enter the stopped state.

#### 12.1.14 Power Down Mode (8048AH, 8049AH, 8050AH, 8039AHL, 8035AHL, 8040AHL)

Extra circuitry has been added to the 8048AH/8049AH/8050AH ROM version to allow power to be removed from all but the data RAM array for low power standby operation. In the power down mode the contents of data RAM can be maintained while drawing typically 10% to 15% of normal operating power requirements.

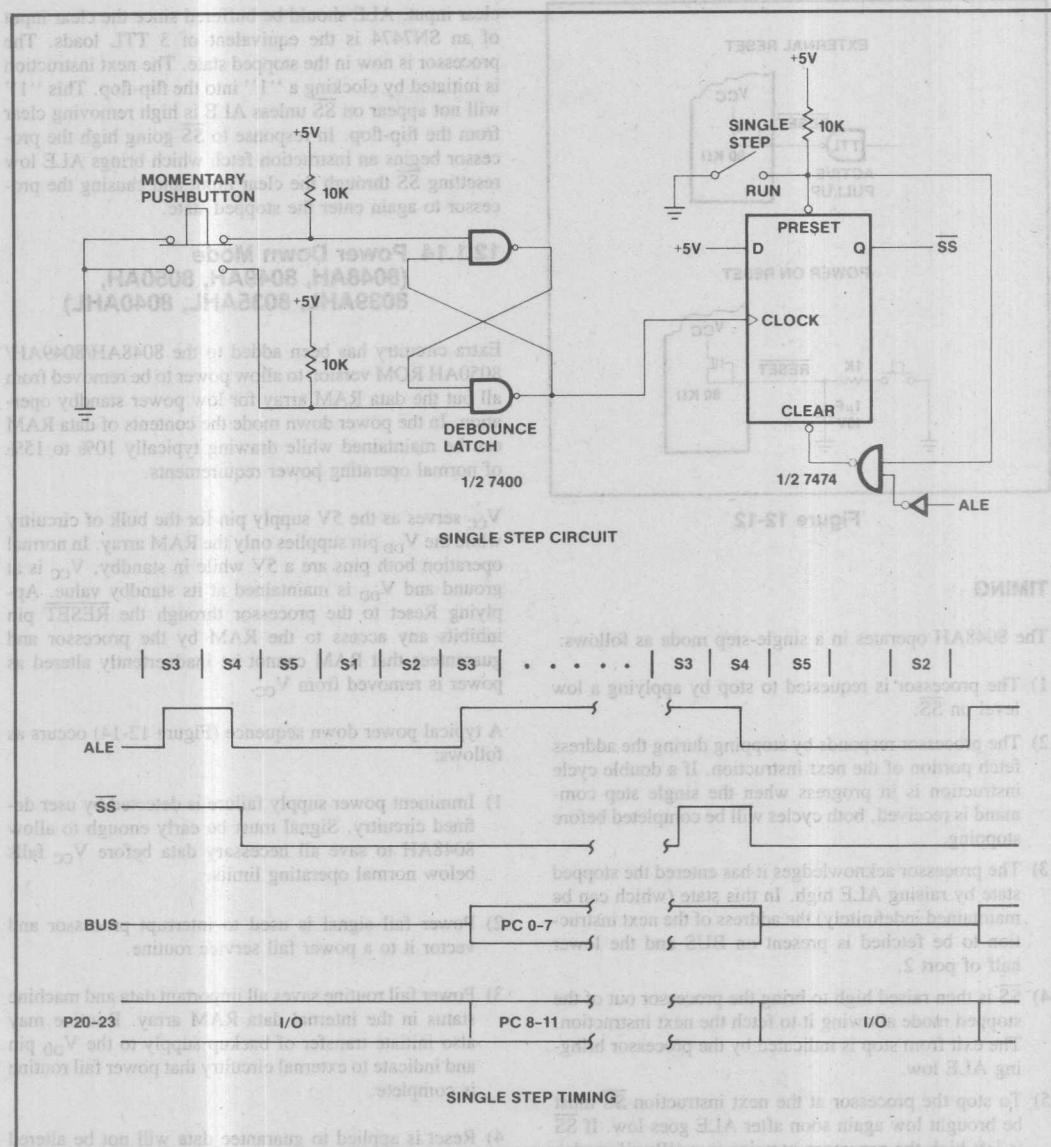
$V_{CC}$  serves as the 5V supply pin for the bulk of circuitry while the  $V_{DD}$  pin supplies only the RAM array. In normal operation both pins are a 5V while in standby,  $V_{CC}$  is at ground and  $V_{DD}$  is maintained at its standby value. Applying Reset to the processor through the  $\overline{RESET}$  pin inhibits any access to the RAM by the processor and guarantees that RAM cannot be inadvertently altered as power is removed from  $V_{CC}$ .

A typical power down sequence (Figure 12-14) occurs as follows:

- 1) Imminent power supply failure is detected by user defined circuitry. Signal must be early enough to allow 8048AH to save all necessary data before  $V_{CC}$  falls below normal operating limits.
- 2) Power fail signal is used to interrupt processor and vector it to a power fail service routine.
- 3) Power fail routine saves all important data and machine status in the internal data RAM array. Routine may also initiate transfer of backup supply to the  $V_{DD}$  pin and indicate to external circuitry that power fail routine is complete.
- 4) Reset is applied to guarantee data will not be altered as the power supply falls out of limits. Reset must be held low until  $V_{CC}$  is at ground level.

Recovery from the Power Down mode can occur as any other power-on sequence with an external capacitor on the Reset input providing the necessary delay. See the previous section on Reset.





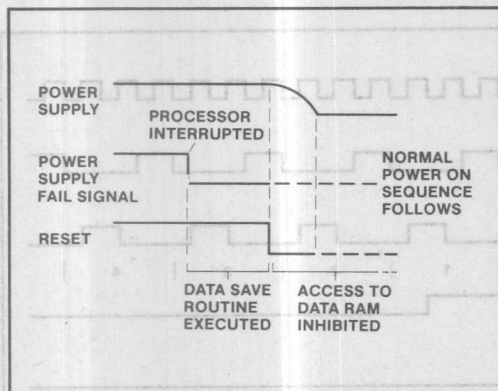


Figure 12-14. Power Down Sequence

### 12.1.15 External Access Mode

Normally the first 1K (8048AH), 2K (8049AH), or 4K (8050AH) words of program memory are automatically fetched from internal ROM or EPROM. The EA input pin however allows the user to effectively disable internal program memory by forcing all program memory fetches to reference external memory. The following chapter explains how access to external program memory is accomplished.

The External Access mode is very useful in system test and debug because it allows the user to disable his internal applications program and substitute an external program of his choice — a diagnostic routine for instance. In addition, section 12.4 explains how internal program memory can be read externally, independent of the processor. A "1" level on EA initiates the external access mode. For proper operation, Reset should be applied while the EA input is changed.

### 12.1.16 Sync Mode

The 8048AH, 8049AH, 8050AH has incorporated a new SYNC mode. The Sync mode is provided to ease the design of multiple controller circuits by allowing the designer to force the device into known phase and state time. The SYNC mode may also be utilized by automatic test equipment (ATE) for quick, easy, and efficient synchronizing between the tester and the DUT (device under test).

SYNC mode is enabled when SS' pin is raised to high voltage level of +12 volts. To begin synchronization, T0 is raised to 5 volts at least four clock cycles after SS'. T0 must be high for at least four X1 clock cycles to fully

reset the prescaler and time state generators. T0 may then be brought down with the rising edge of X1. Two clock cycles later, with the rising edge of X1, the device enters into Time State 1, Phase 1. SS' is then brought down to 5 volts 4 clocks later after T0. RESET' is allowed to go high 5 tCY (75 clocks) later for normal execution of code. See Figure 12-15.

### 12.1.17 Idle Mode

Along with the standard power down, the 80C438, 80C49, 80C50 has added an IDLE mode instruction (01H) to give even further flexibility and power management. In the IDLE mode, the CPU is frozen while the oscillator, RAM, timer, and the interrupt circuitry remains fully active.

When the IDL instruction (01H) is decoded, the clock to the CPU is stopped. CPU status is preserved in its entirety: the Stack Pointer, Program Counter, Program Status Word, Accumulator, RAM, and all the registers maintain their data throughout idle.

Externally, the following occurs during idle:

- 1) The ports remain in the logical state they were in when idle was executed.
- 2) The bus remains in the logical state it was in when idle was executed if the bus was latched.  
If the bus was in a high Z condition or if external program memory is used the bus will remain in the float state.
- 3) ALE remains in the inactive state (low).
- 4) RD', WR', PROG', and PSEN' remains in the inactive state (high).
- 5) T0 outputs clock if enabled.

There are three ways of exiting idle. Activating any enabled interrupt (external or timer) will cause the CPU to vector to the appropriate interrupt routine. Following a RETR instruction, program execution will resume at the instruction following the address that contained the IDL instruction.

The F0 and F1 flags may be used to give an indication if the interrupt occurred during normal program execution or during idle. This is done by setting or clearing the flags before going into idle. The interrupt service routine can examine the flags and act accordingly when idle is terminated by an interrupt.

Resetting the device can also terminate idle. Since the oscillator is already running, five machine cycles are all that is required to insure proper machine operation.

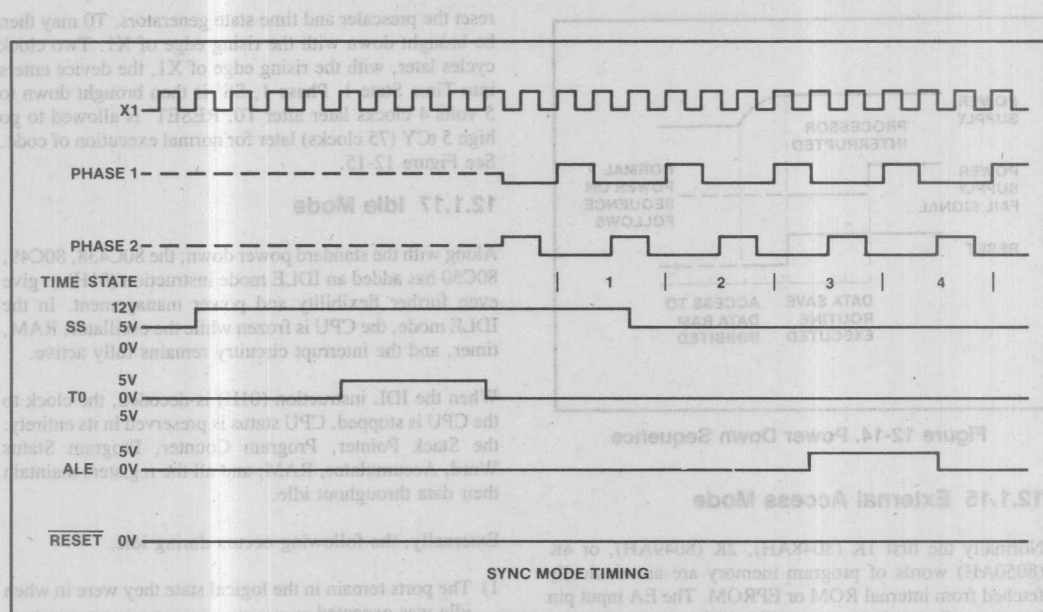


Figure 12-15. Sync Mode Timing

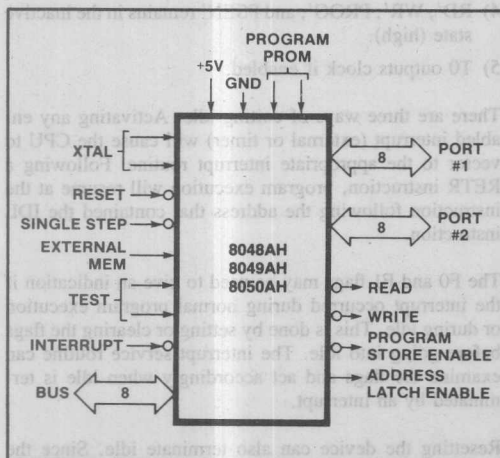


Figure 12-16. 8048AH and 8049AH Logic Symbol

## 12.2 PIN DESCRIPTION

The MCS-48 processors are packaged in 40 pin Dual In-Line Packages (DIP's). Table 12-3 is a summary of the functions of each pin. Figure 12-16 is the logic symbol for the 8048AH product family. Where it exists, the second paragraph describes each pin's function in an expanded MCS-48 system. Unless otherwise specified, each input is TTL compatible and each output will drive one standard TTL load.

**XTAL** (pin 14) is the crystal input. It is used to provide a clock signal to the processor. The XTAL pin is connected to a crystal and the other end of the crystal is connected to ground. The XTAL pin is also connected to the XTAL pin of the 8049AH and 8050AH processors.

# SINGLE COMPONENT MCS®-48 SYSTEM

Table 12-3. Pin Description

Designation	Pin Number*	Function
V <sub>SS</sub>	20	Circuit GND potential
V <sub>DD</sub>	26	Programming power supply; 21V during program for the 8748H/8749H; +5V during operation for both ROM and EPROM. Low power standby pin in 8048AH and 8049AH/8050AH ROM versions.
V <sub>CC</sub>	40	Main power supply; +5V during operation and during 8748H and 8749H programming.
PROG	25	Program pulse; +18V input pin during 8748H/8749H programming. Output strobe for 8243 I/O expander.
P10-P17 (Port 1)	27-34	8-bit quasi-bidirectional port. (Internal Pullup $\approx$ 50K $\Omega$ )
P20-P27 (Port 2)	21-24 35-38	8-bit quasi-bidirectional port. (Internal Pullup $\approx$ 50K $\Omega$ ) P20-P23 contain the four high order program counter bits during an external program memory fetch and serve as a 4-bit I/O expander bus for 8243.
D0-D7 (BUS)	12-19	True bidirectional port which can be written or read synchronously using the $\overline{RD}$ , $\overline{WR}$ strobes. The port can also be statically latched. Contains the 8 low order program counter bits during an external program memory fetch, and receives the addressed instruction under the control of PSEN. Also contains the address and data during an external RAM data store instruction, under control of ALE, $\overline{RD}$ , and $\overline{WR}$ .
T0	1	Input pin testable using the conditional transfer instructions JT0 and JNT0. T0 can be designated as a clock output using ENTO CLK instruction. T0 is also used during programming and sync mode.
TI	39	Input pin testable using the JT1, and JNT1 instructions. Can be designated the event counter input using the STRT CNT instruction. (See Section 2.1.10)
INT	6	Interrupt input. Initiates an interrupt if interrupt is enabled. Interrupt is disabled after a reset. (Active low) Interrupt must remain low for at least 3 machine cycles to ensure proper operation.
$\overline{RD}$	8	Output strobe activated during a BUS read. Can be used to enable data onto the BUS from an external device. (Active low) Used as a Read Strobe to External Data Memory.
$\overline{RESET}$	4	Input which is used to initialize the processor. Also used during EPROM programming and verification. (Active low) (Internal pullup $\approx$ 80K $\Omega$ )
$\overline{WR}$	10	Output strobe during a BUS write. (Active low) Used as write strobe to external data memory.
ALE	11	Address Latch Enable. This signal occurs once during each cycle and is useful as a clock output. The negative edge of ALE strobes address into external data and program memory.



Table 12-3. Pin Description (Continued)

Designation	Pin Number*	Function
PSEN	9	Program Store Enable. This output occurs only during a fetch to external program memory. (Active low)
SS	5	Single step input can be used in conjunction with ALE to "single step" the processor through each instruction. (Active low) (Internal pullup $\approx 300K\Omega$ ) +12V for sync modes (See 2.1.16)
EA	7	External Access input which forces all program memory fetches to reference external memory. Useful for emulation and debug, and essential for testing and program verification. (Active high) +12V for 8048AH/8049AH/8050AH program verification and +18V for 8748H/8749H program verification (Internal pullup $\approx 10M\Omega$ on 8048AH/8049AH/8035AHL/8039AHL/8050AH/8040AHL)
XTAL1	2	One side of crystal input for internal oscillator. Also input for external source.
XTAL2	3	Other side of crystal/external source input.

\*Unless otherwise stated, inputs do not have internal pullup resistors. 8048AH, 8748H, 8049AH, 8050AH, 8040AHL

## 12.3 PROGRAMMING, VERIFYING AND ERASING EPROM

The internal Program Memory of the 8748H and the 8749H may be erased and reprogrammed by the user as explained in the following sections. See also the 8748H and 8749H data sheets.

### 12.3.1 Programming/Verification

In brief, the programming process consists of: activating the program mode, applying an address, latching the address, applying data, and applying a programming pulse. This programming algorithm applies to both the 8748H and 8749H. Each word is programmed completely before moving on to the next and is followed by a verification step. The following is a list of the pins used for programming and a description of their functions:

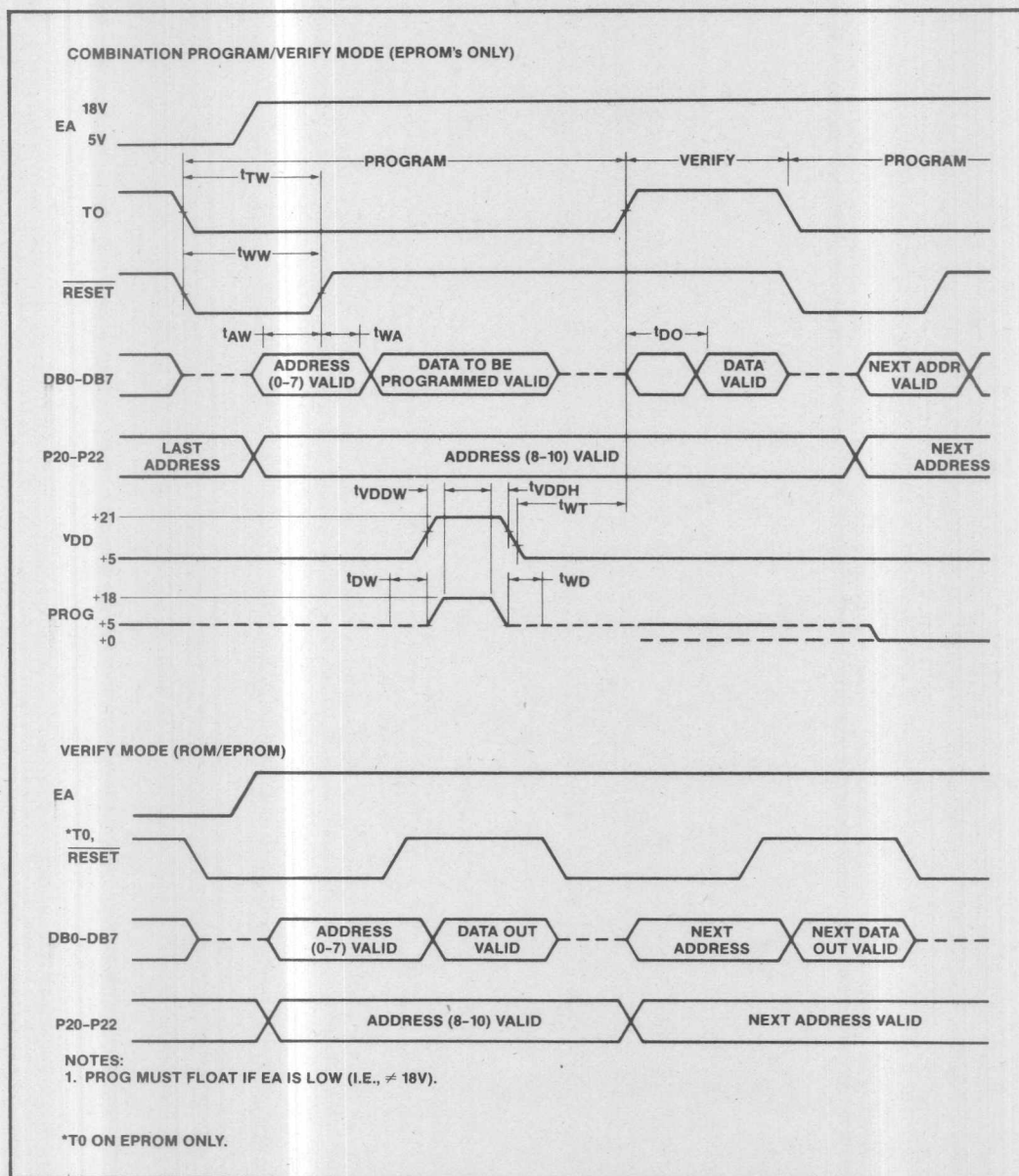
Pin	Function
XTAL1	Clock Input (3 to 4 MHz)
Reset	Initialization and Address Latching
Test 0	Selection of Program (0V) or Verify (5V) Mode
EA	Activation of Program/Verify Modes
BUS	Address and Data Input Data Output During Verify
P20-1	Address Input for 8748H
P20-2	Address Input for 8749H
V <sub>DD</sub>	Programming Power Supply
PROG	Program Pulse Input
P10-P11	Tied to ground (8749H only)

## 8748H AND 8749H ERASURE CHARACTERISTICS

The erasure characteristics of the 8748H and 8749H are such that erasure begins to occur when exposed to light with wavelengths shorter than approximately 4000 Angstroms (Å). It should be noted that sunlight and certain types of fluorescent lamps have wavelengths in the 3000-4000Å range. Data show that constant exposure to room level fluorescent lighting could erase the typical 8748H and 8749H in approximately 3 years while it would take approximately 1 week to cause erasure when exposed to direct sunlight. If the 8748H or 8749H is to be exposed to these types of lighting conditions for extended periods of time, opaque labels should be placed over the 8748H window to prevent unintentional erasure.

When erased, bits of the 8748H and 8749H Program Memory are in the logic "0" state.

The recommended erasure procedure for the 8748H and 8749H is exposure to shortwave ultraviolet light which has a wavelength of 2537 Angstroms (Å). The integrated dose (i.e., UV intensity X exposure time) for erasure should be a minimum of 15W-sec/cm<sup>2</sup>. The erasure time with this dosage is approximately 15 to 20 minutes using an ultraviolet lamp with a 12000μW/cm<sup>2</sup> power rating. The 8748H and 8749H should be placed within one inch from the lamp tubes during erasure. Some lamps have a filter in their tubes and this filter should be removed before erasure.



**Figure 12-17. Program/Verify Sequence for 8749H/8748H**

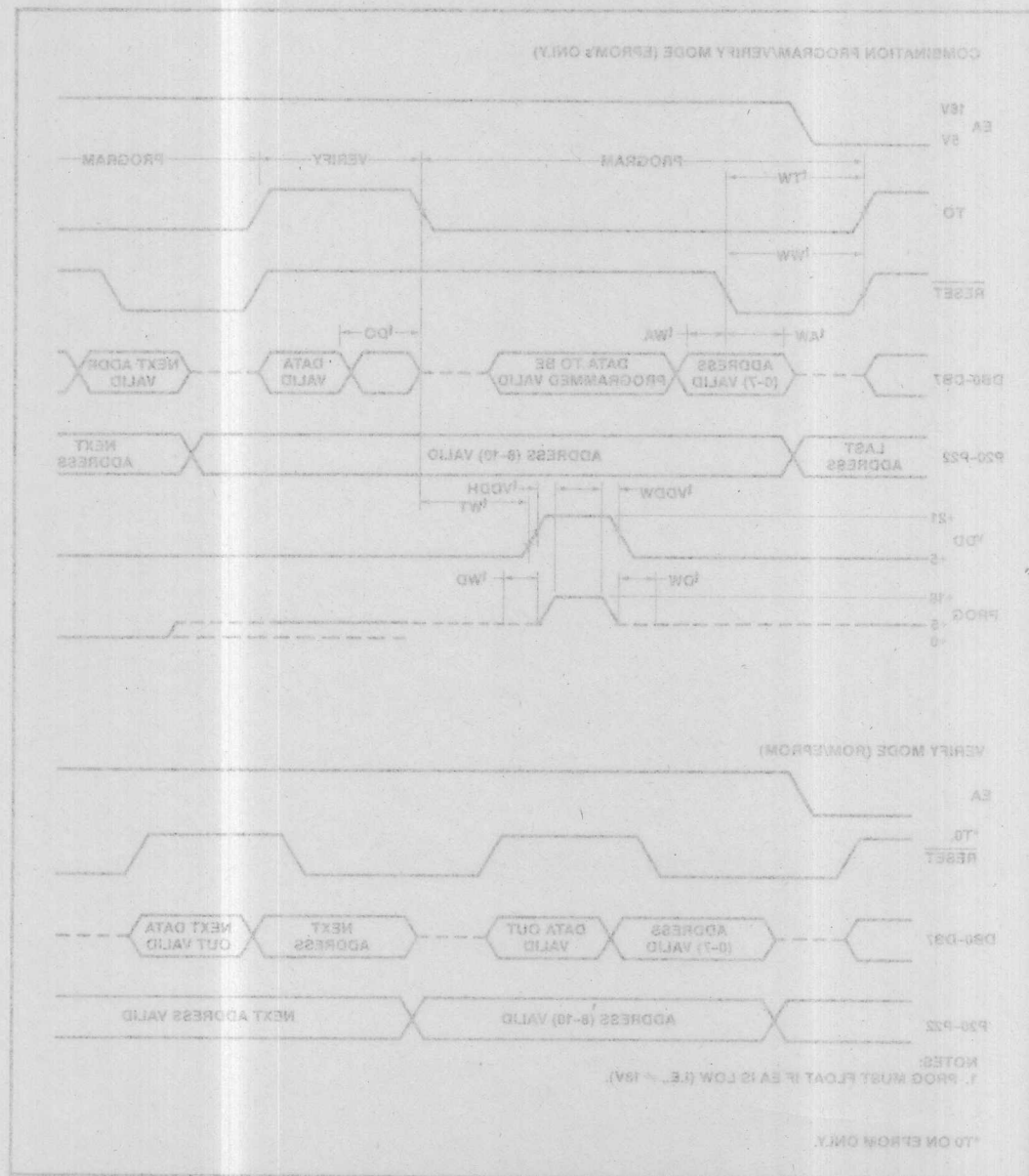


Figure 12-17. Program/Verify Sequence for 8748H/8748H







## CHAPTER 13 EXPANDED MCS®-48 SYSTEM

### 13.0 INTRODUCTION

If the capabilities resident on the single-chip 8048AH/8748H/8035AHL/8049AH/8749H/8039AHL are not sufficient for your system requirements, special on-board circuitry allows the addition of a wide variety of external memory, I/O, or special peripherals you may require. The processors can be directly and simply expanded in the following areas:

- Program Memory to 4K words
- Data Memory to 320 words (384 words with 8049AH)
- I/O by unlimited amount
- Special Functions using 8080/8085AH peripherals

By using bank switching techniques, maximum capability is essentially unlimited. Bank switching is discussed later in the chapter. Expansion is accomplished in two ways:

- 1) Expander I/O — A special I/O Expander circuit, the 8243, provides for the addition of four 4-bit Input/Output ports with the sacrifice of only the lower half (4-bits) of port 2 for inter-device communication. Multiple 8243's may be added to this 4-bit bus by generating the required "chip select" lines.
- 2) Standard 8085 Bus — One port of the 8048AH/8049AH is like the 8-bit bidirectional data bus of the 8085 microcomputer system allowing interface to the numerous standard memories and peripherals of the MCS®-80/85 microcomputer family.

MCS-48 systems can be configured using either or both of these expansion features to optimize system capabilities to the application.

Both expander devices and standard memories and peripherals can be added in virtually any number and combination required.

### 13.1 EXPANSION OF PROGRAM MEMORY

Program Memory is expanded beyond the resident 1K or 2K words by using the 8085 BUS feature of the MCS®-48. All program memory fetches from the addresses less than 1024 on the 8048AH and less than 2048 on the 8049AH occur internally with no external signals being generated (except ALE which is always present). At address 1024 on the 8048AH, the processor automatically initiates external program memory fetches.

#### 13.1.1 Instruction Fetch Cycle (External)

As shown in Figure 13-1, for all instruction fetches from addresses of 1024 (2048) or greater, the following will occur:

- 1) The contents of the 12-bit program counter will be output on BUS and the lower half of port 2.
- 2) Address Latch Enable (ALE) will indicate the time at which address is valid. The trailing edge of ALE is used to latch the address externally.
- 3) Program Store Enable (PSEN) indicates that an external instruction fetch is in progress and serves to enable the external memory device.
- 4) BUS reverts to input (floating) mode and the processor accepts its 8-bit contents as an instruction word.

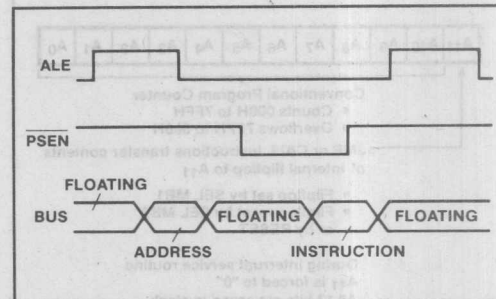


Figure 13-1. Instruction Fetch from External Program Memory

All instruction fetches, including internal addresses, can be forced to be external by activating the EA pin of the 8048AH/8049AH/8050AH. The 8035AHL/8039AHL/8040AHL processors without program memory always operate in the external program memory mode (EA = 5V).

#### 13.1.2 Extended Program Memory Addressing (Beyond 2K)

For programs of 2K words or less, the 8048AH/8049AH addresses program memory in the conventional manner. Addresses beyond 2047 can be reached by executing a program memory bank switch instruction (SEL MB0, SEL MB1) followed by a branch instruction (JMP or CALL). The bank switch feature extends the range of branch instructions beyond their normal 2K range and at the same time prevents the user from inadvertently crossing the 2K boundary.

#### PROGRAM MEMORY BANK SWITCH

The switching of 2K program memory banks is accomplished by directly setting or resetting the most significant bit of the program counter (bit 11); see Figure 13-2. Bit 11 is not altered by normal incrementing of the program counter but is loaded with the contents of a special flip-flop each time a JMP or CALL instruction is executed. This special flip-flop is set by executing an SEL MB1

instruction and reset by SEL MB0. Therefore, the SEL MB instruction may be executed at any time prior to the actual bank switch which occurs during the next branch instruction encountered. Since all twelve bits of the program counter, including bit 11, are stored in the stack, when a Call is executed, the user may jump to subroutines across the 2K boundary and the proper bank will be restored upon return. However, the bank switch flip-flop will not be altered on return.

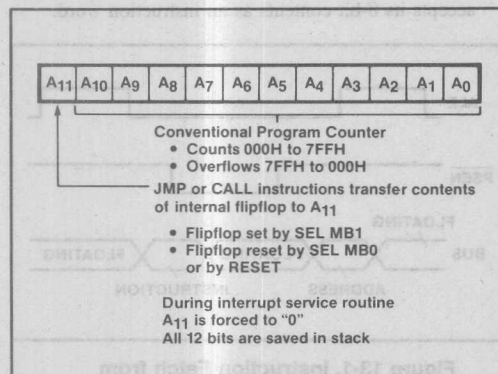


Figure 13-2. Program Counter

## INTERRUPT ROUTINES

Interrupts always vector the program counter to location 3 or 7 in the first 2K bank, and bit 11 of the program

counter is held at "0" during the interrupt service routine. The end of the service routine is signalled by the execution of an RETR instruction. Interrupt service routines should therefore be contained entirely in the lower 2K words of program memory. The execution of a SEL MB0 or SEL MB1 instruction within an interrupt routine is not recommended since it will not alter PC11 while in the routine, but will change the internal flip-flop.

## 13.1.3 Restoring I/O Port Information

Although the lower half of Port 2 is used to output the four most significant bits of address during an external program memory fetch, the I/O information is still output during certain portions of each machine cycle. I/O information is always present on Port 2's lower 4 bits at the rising edge of ALE and can be sampled or latched at this time.

## 13.1.4 Expansion Examples

Shown in Figure 13-3 is the addition of 2K words of program memory using an 2716A 2K x 8 ROM to give a total of 3K words of program memory. In this case no chip select decoding is required and PSEN enables the memory directly through the chip select input. If the system requires only 2K of program memory, the same configuration can be used with an 8035AHL substituted for the 8048AH. The 8049AH would provide 4K of program memory with the same configuration.

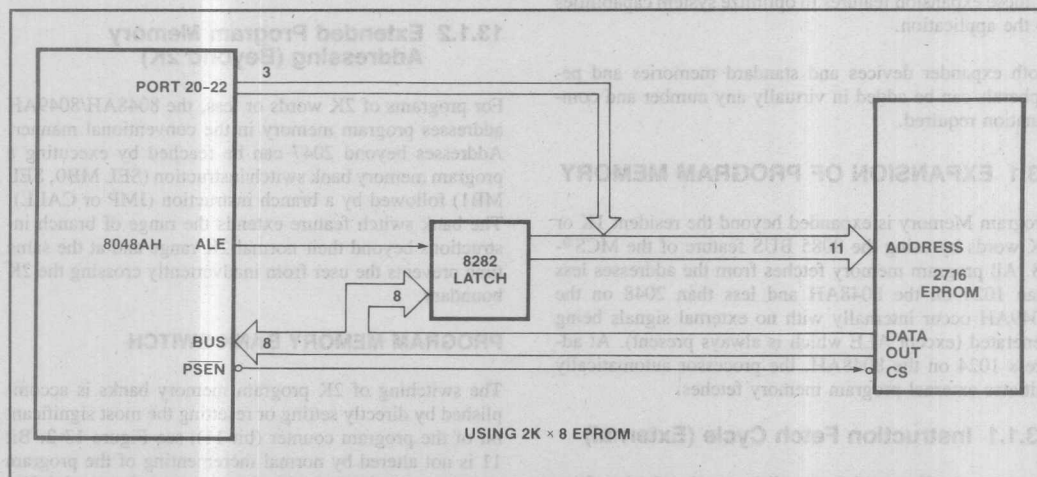


Figure 13-3. Expanding MCS-48 Program Memory Using Standard Memory Products

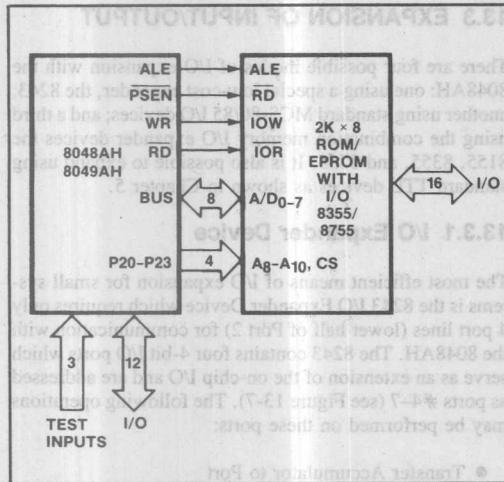


Figure 13-4. External Program Memory Interface

Figure 13-4 shows how the 8755/8355 EPROM/ROM with I/O interfaces directly to the 8048AH without the need for an address latch. The 8755/8355 contains an internal 8-bit address latch eliminating the need for an 8212 latch. In addition to a 2K x 8 program memory, the 8755/8355 also contains 16 I/O lines addressable as two 8-bit ports. These ports are addressed as external RAM; therefore the  $\overline{RD}$  and  $\overline{WR}$  outputs of the 8048AH are required. See the following section on data memory expansion for more detail. The subsequent section on I/O expansion explains the operation of the 16 I/O lines.

### 13.2 EXPANSION OF DATA MEMORY

Data Memory is expanded beyond the resident 64 words by using the 8085AH type bus feature of the MCS®-48.

#### 13.2.1 Read/Write Cycle

All address and data is transferred over the 8 lines of BUS. As shown in Figure 13-5, a read or write cycle occurs as follows:

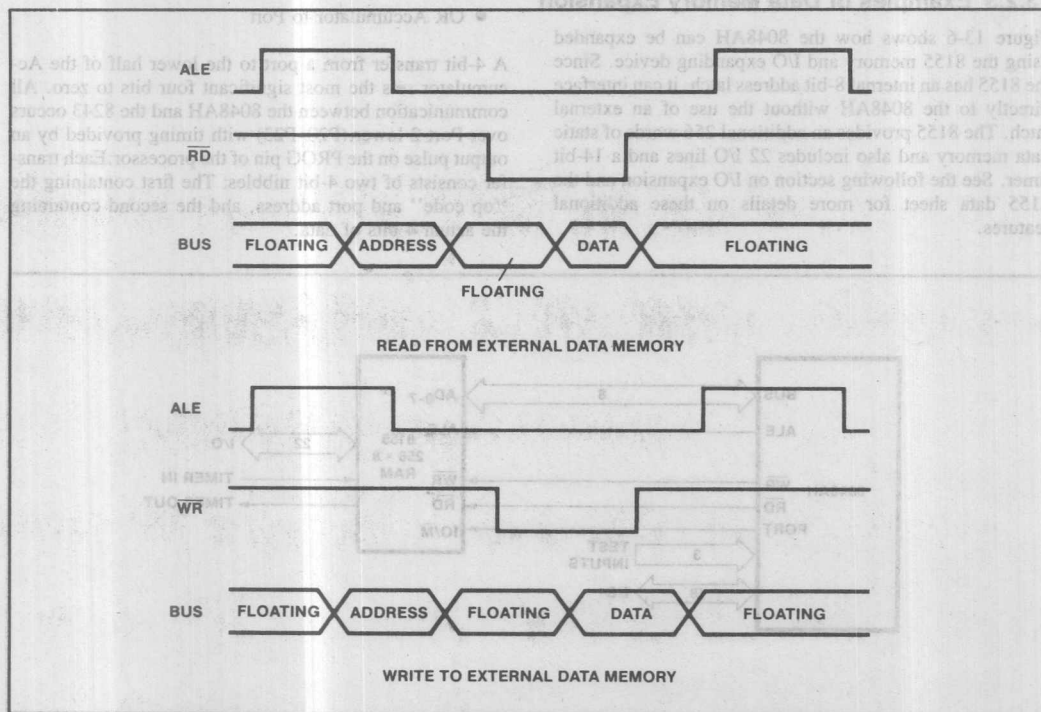


Figure 13-5. External Data Memory Timings



- 1) The contents of register R0 or R1 is outputted on BUS.
- 2) Address Latch Enable (ALE) indicates address is valid. The trailing edge of ALE is used to latch the address externally.
- 3) A read ( $\overline{RD}$ ) or write ( $\overline{WR}$ ) pulse on the corresponding output pins of the 8048AH indicates the type of data memory access in progress. Output data is valid at the trailing edge of  $\overline{WR}$  and input data must be valid at the trailing edge of  $\overline{RD}$ .
- 4) Dat (8 bits) is transferred in or out over BUS.

### 13.2.2 Addressing External Data Memory

External Data Memory is accessed with its own two-cycle move instructions, MOVXA, @R and MOVX@R, A, which transfer 8 bits of data between the accumulator and the external memory location addressed by the contents of one of the RAM Pointer Registers R0 and R1. This allows 256 locations to be addressed in addition to the resident locations. Additional pages may be added by "bank switching" with extra output lines of the 8048AH.

### 13.2.3 Examples of Data Memory Expansion

Figure 13-6 shows how the 8048AH can be expanded using the 8155 memory and I/O expanding device. Since the 8155 has an internal 8-bit address latch, it can interface directly to the 8048AH without the use of an external latch. The 8155 provides an additional 256 words of static data memory and also includes 22 I/O lines and a 14-bit timer. See the following section on I/O expansion and the 8155 data sheet for more details on these additional features.

### 13.3 EXPANSION OF INPUT/OUTPUT

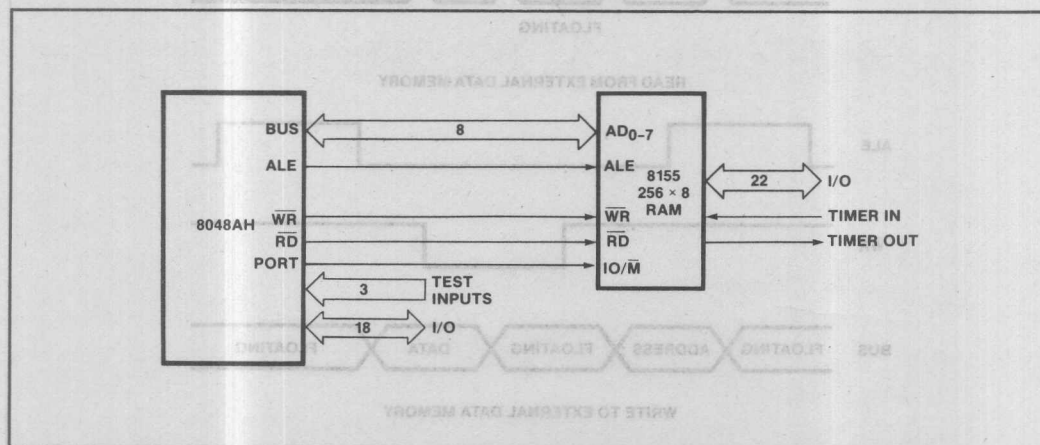
There are four possible modes of I/O expansion with the 8048AH: one using a special low-cost expander, the 8243; another using standard MCS-80/85 I/O devices; and a third using the combination memory I/O expander devices the 8155, 8355, and 8755. It is also possible to expand using standard TTL devices as shown in Chapter 5.

### 13.3.1 I/O Expander Device

The most efficient means of I/O expansion for small systems is the 8243 I/O Expander Device which requires only 4 port lines (lower half of Port 2) for communication with the 8048AH. The 8243 contains four 4-bit I/O ports which serve as an extension of the on-chip I/O and are addressed as ports #4-7 (see Figure 13-7). The following operations may be performed on these ports:

- Transfer Accumulator to Port
- Transfer Port to Accumulator
- AND Accumulator to Port
- OR Accumulator to Port

A 4-bit transfer from a port to the lower half of the Accumulator sets the most significant four bits to zero. A communication between the 8048AH and the 8243 occurs over Port 2 lower (P20–P23) with timing provided by an output pulse on the PROG pin of the processor. Each transfer consists of two 4-bit nibbles: The first containing the “op code” and port address, and the second containing the actual 4 bits of data.



**Figure 13-6. 8048AH Interface to 256 x 8 Standard Memories**

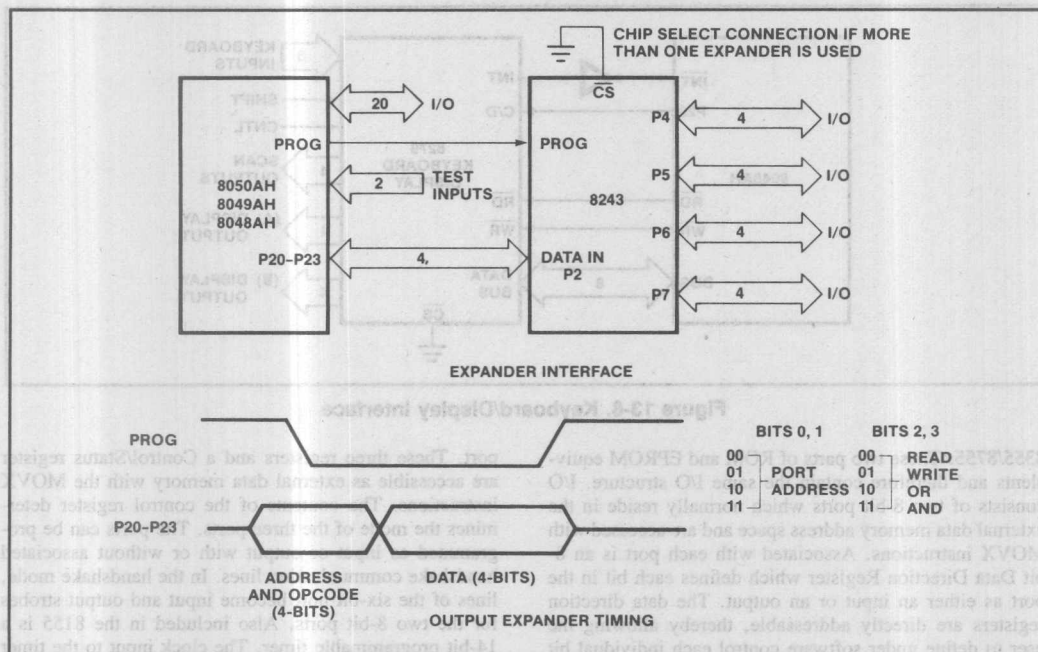
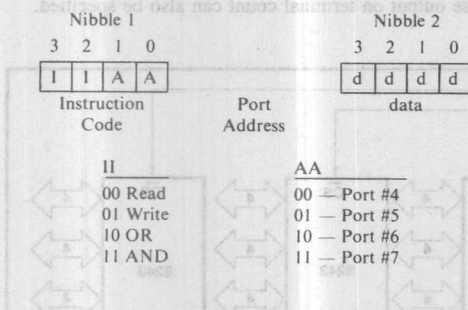


Figure 13-7. 8243 Expander I/O Interface



A high to low transition of the PROG line indicates that address is present, while low to high transition indicates the presence of data. Additional 8243's may be added to the four-bit bus and chip selected using additional output lines from the 8048AH/8748H.

#### I/O PORT CHARACTERISTICS

Each of the four 4-bit ports of the 8243 can serve as either input or output and can provide high drive capability in both the high and low state.

#### 13.3.2 I/O Expansion with Standard Peripherals

Standard MCS-80/85 type I/O devices may be added to the MCS®-48 using the same bus and timing used for Data Memory expansion. Figure 13-8 shows an example of how an 8048AH can be connected to an MCS-85 peripheral. I/O devices reside on the Data Memory bus and in the data memory address space and are accessed with the same MOVX instructions. (See the previous section on data memory expansion for a description of timing.) The following are a few of the Standard MCS-80 devices which are very useful in MCS®-48 systems:

- 8214 Priority Interrupt Encoder
- 8251 Serial Communications Interface
- 8255 General Purpose Programmable I/O
- 8279 Keyboard/Display Interface
- 8253 Interval Timer

#### 13.3.3 Combination Memory and I/O Expanders

As mentioned in the sections on program and data memory expansion, the 8355/8755 and 8155 expanders also contain I/O capability.

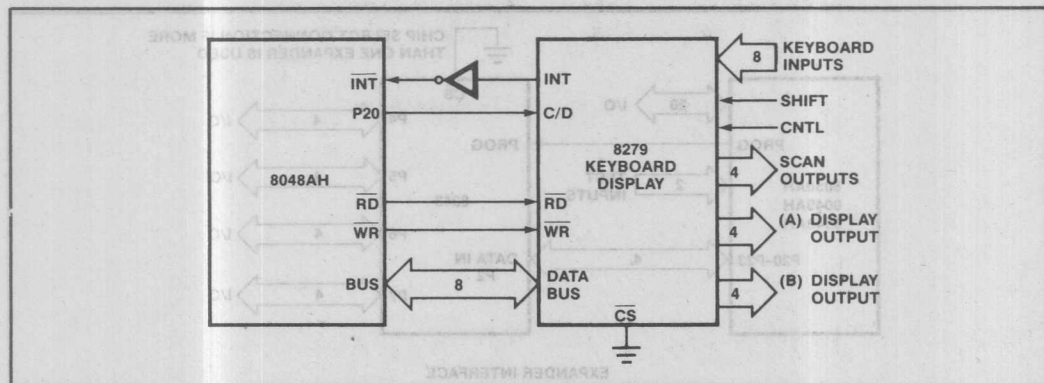


Figure 13-8. Keyboard/Display Interface

**8355/8755:** These two parts of ROM and EPROM equivalents and therefore contain the same I/O structure. I/O consists of two 8-bit ports which normally reside in the external data memory address space and are accessed with MOVX instructions. Associated with each port is an 8-bit Data Direction Register which defines each bit in the port as either an input or an output. The data direction registers are directly addressable, thereby allowing the user to define under software control each individual bit of the ports as either input or output. All outputs are statically latched and double buffered. Inputs are not latched.

**8155/8156:** I/O on the 8155/8156 is configured as two 8-bit programmable I/O ports and one 6-bit programmable

port. These three registers and a Control/Status register are accessible as external data memory with the MOVX instructions. The contents of the control register determines the mode of the three ports. The ports can be programmed as input or output with or without associated handshake communication lines. In the handshake mode, lines of the six-bit port become input and output strobes for the two 8-bit ports. Also included in the 8155 is a 14-bit programmable timer. The clock input to the timer and the timer overflow output are available on external pins. The timer can be programmed to stop on terminal count or to continuously reload itself. A square wave or pulse output on terminal count can also be specified.

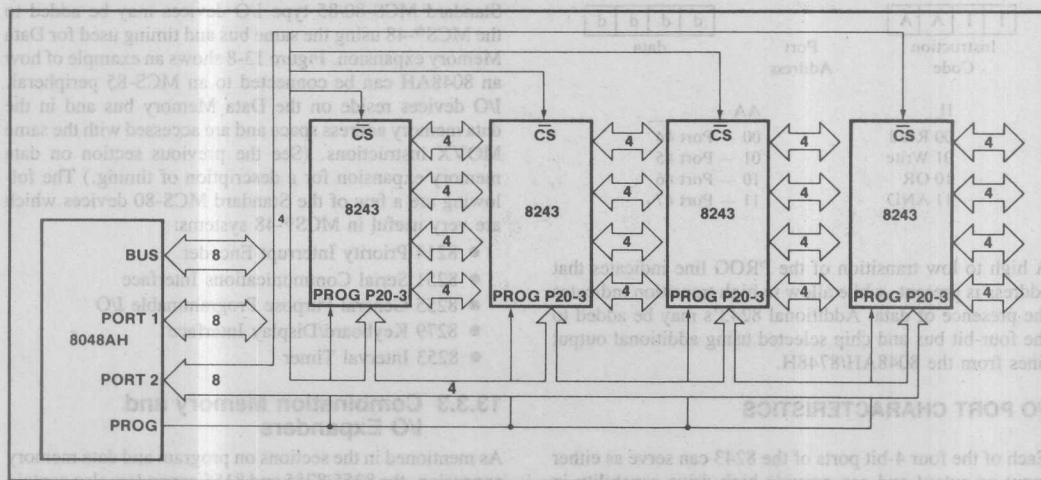


Figure 13-9. Low Cost I/O Expansion

## I/O EXPANSION EXAMPLES

Figure 13-9 shows the expansion of I/O using multiple 8243's. The only difference from a single 8243 system is the addition of chip selects provided by additional 8048AH output lines. Two output lines and a decoder could also be used to address the four chips. Large numbers of 8243's would require a chip select decoder chip such as the 8205 to save I/O pins.

Figure 13-10 shows the 8048AH interface to a standard MCS<sup>®</sup>-80 peripheral; in this case, the 8255 Programmable Peripheral Interface, a 40-pin part which provides three 8-bit programmable I/O ports. The 8255 bus interface is typical of programmable MCS<sup>®</sup>-80 peripherals with an 8-bit bidirectional data bus, a RD and WR input for Read/Write control, a CS (chip select) input used to enable the Read/Write control logic and the address inputs used to select various internal registers.

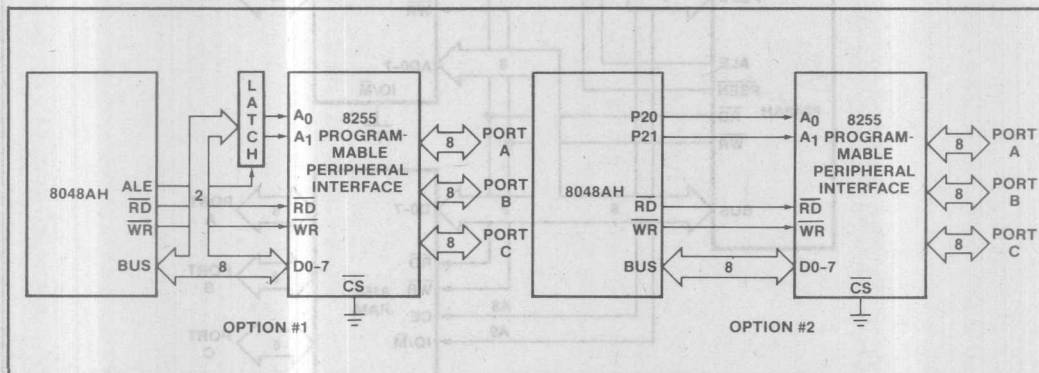


Figure 13-10. Interface to MCS<sup>®</sup>-80 Peripherals

Interconnection to the 8048AH is very straightforward with BUS, RD, and WR connecting directly to the corresponding pins on the 8255. The only design consideration is the way in which the internal registers of the 8255 are to be addressed. If the registers are to be addressed as external data memory using the MOVX instructions, the appropriate number of address bits (in this case, 2) must be latched on BUS using ALE as described in the section on external data memories. If only a single device is connected to BUS, the 8255 may be continuously selected by grounding  $\overline{CS}$ . If multiple 8255's are used, additional address bits can be latched and used as chip selects.

A second addressing method eliminates external latches and chip select decoders by using output port lines as address and chip select lines directly. This method, of course, requires the setting of an output port with address information prior to executing a MOVX instruction.

### 13.4 MULTI-CHIP MCS<sup>®</sup>-48 SYSTEMS

Figure 13-11 shows the addition of two memory expanders to the 8048AH, one 8355/8755 ROM and one 8156 RAM. The main consideration in designing such a system is the

addressing of the various memories and I/O ports. Note that in this configuration address lines  $A_{10}$  and  $A_{11}$  have been tied to chip select the 8355. This ensures that the chip is active for all external program memory fetches in the 1K to 3K range and is disabled for all other addresses. This gating has been added to allow the I/O port of the 8355 to be used. If the chip was left selected all the time, there would be conflict between these ports and the RAM and I/O of the 8156. The NOR gate could be eliminated and  $A_{11}$  connected directly to the CE (instead of  $\overline{CE}$ ) input of the 8355; however, this would create a 1K word "hole" in the program memory by causing the 8355 to be active in the 2K and 4K range instead of the normal 1K to 3K range.

In this system the various locations are addressed as follows:

- Data RAM — Addresses 0 to 255 when Port 2 Bit 0 has been previously set = 1 and Bit 1 set = 0
- RAM I/O — Addresses 0 to 3 when Port 2 Bit 0 = 1 and Bit 1 = 1
- ROM I/O — Addresses 0 to 3 when Port 2 Bit 2 or Bit 3 = 1

See the memory map in Figure 13-12.



## Component MCS®-48 System

### 13.5 MEMORY BANK SWITCHING

• Data RAM — Address 0 to 255 when Port 2 Bit 1 is set.

another bit of the program counter.

to activate the proper bank.

Table 12 summarizes the information that is obtained from the

active state.

Table 13-1. MCS®-48 Control Signals

Control Signal	When Active
$\overline{RD}$	During MOVX, A, @R, or INS Bus
$\overline{WR}$	During MOVX @R, A or OUTL Bus
ALE	Every Machine Cycle
$\overline{PSEN}$	During Fetch of external program memory (instruction or immediate data)
PROG	During MOVD, A,P ANLD P,A MOVD P,A ORLD P,A

### 13.7 PORT CHARACTERISTICS

#### 13.7 BUS Port Operations

The BUS port can operate in three different modes: as a latched I/O port, as a bidirectional bus port, or as a program memory address output when external memory is used. The BUS port lines are either active high, active low, or high impedance (floating).

The latched mode (INS, OUTL) is intended for use in the single-chip configuration where BUS is not begin used as an expander port. OUTL and MOVX instructions can be mixed if necessary. However, a previously latched output will be destroyed by executing a MOVX instruction and BUS will be left in the high impedance state. INS does not put the BUS in a high impedance state. Therefore, the use of MOVX after OUTL to put the BUS in a high impedance state is necessary before an INS instruction intended to read an external word (as opposed to the previously latched value).

OUTL should never be used in a system with external program memory, since latching BUS can cause the next instruction, if external, to be fetched improperly.

#### 13.7.2 Port 2 Operations

The lower half of Port 2 can be used in three different ways: as a quasi-bidirectional static port, as an 8243 expander port, and to address external program memory.

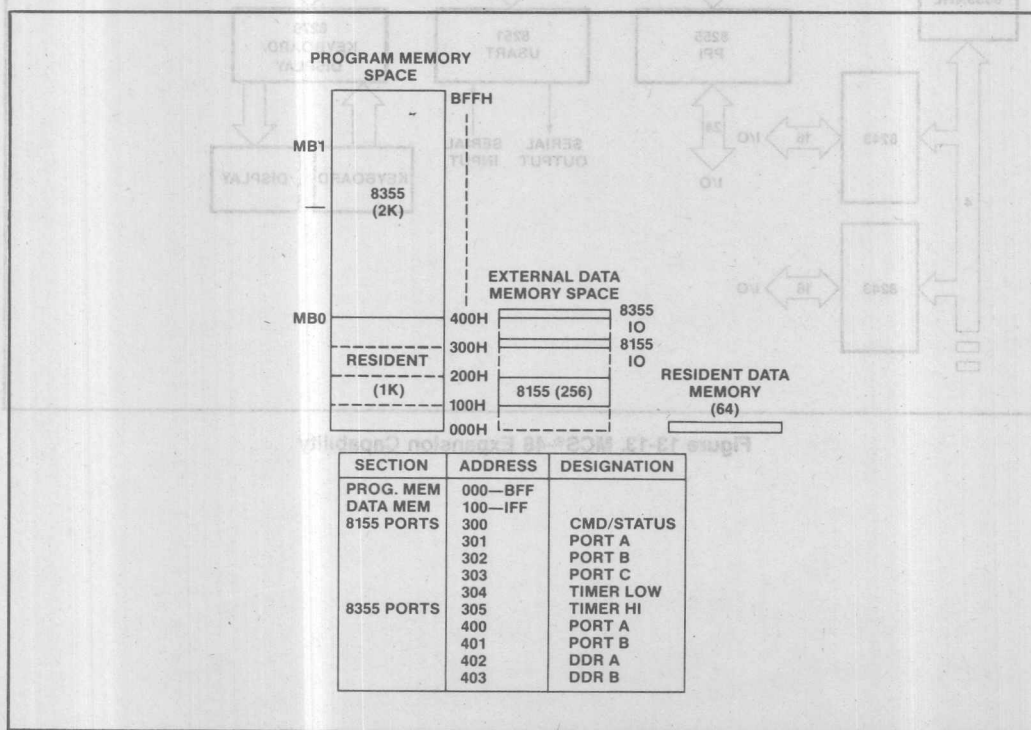


Figure 13-12. Memory Map for Three-Component MCS®-48 Family

## EXPANDED MCS®-48 SYSTEM

In all cases outputs are driven low by an active device and driven high momentarily by a low impedance device and held high by a high impedance device to VCC.

The port may contain latched I/O data prior to its use in another mode without affecting operation of either. If lower Port 2 (P20-3) is used to output address for an external program memory fetch, the I/O information pre-

viously latched will be automatically removed temporarily while address is present, then restored when the fetch is complete. However, if lower Port 2 is used to communicate with an 8243, previously latched I/O information will be removed and not restored. After an input from the 8243, P20-3 will be left in the input mode (floating). After an output to the 8243, P20-3 will contain the value written, ANDed, or ORed to the 8243 port.

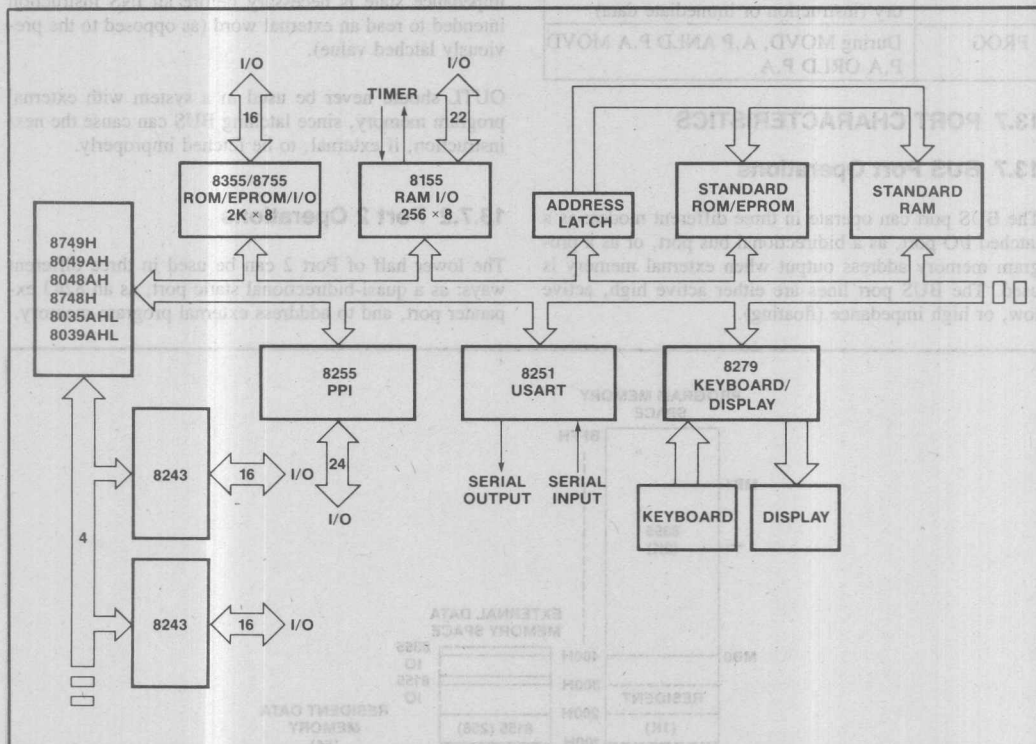


Figure 13-13. MCS®-48 Expansion Capability





14

MCS® 48 Instruction Set

# CHAPTER 14

## MCS®-48 INSTRUCTION SET

### 14.0 INTRODUCTION

The MCS®-48 instruction set is extensive for a machine of its size and has been tailored to be straightforward and very efficient in its use of program memory. All instructions are either one or two bytes in length and over 80% are only one byte long. Also, all instructions execute in either one or two cycles and over 50% of all instructions execute in a single cycle. Double cycle instructions include all immediate instructions, and all I/O instructions.

The MCS-48 microcomputers have been designed to handle arithmetic operations efficiently in both binary and BCD as well as handle the single-bit operations required in control applications. Special instructions have also been included to simplify loop counters, table look-up routines, and N-way branch routines.

### 14.0.1 Data Transfers

As can be seen in Figure 14.1, the 8-bit accumulator is the central point for all data transfers within the 8048. Data can be transferred between the 8 registers of each working register bank and the accumulator directly, i.e., the source or destination register is specified by the instruction. The remaining locations of the internal RAM array are referred to as Data Memory and are addressed indirectly via an address stored in either R0 or R1 of the active register bank. R0 and R1 are also used to indirectly address external data memory when it is present. Transfers to and from internal RAM require one cycle, while transfers to external RAM require two. Constants stored in Program Memory can be loaded directly to the accumulator and to the 8 working registers. Data can also be transferred directly between the accumulator and the on-

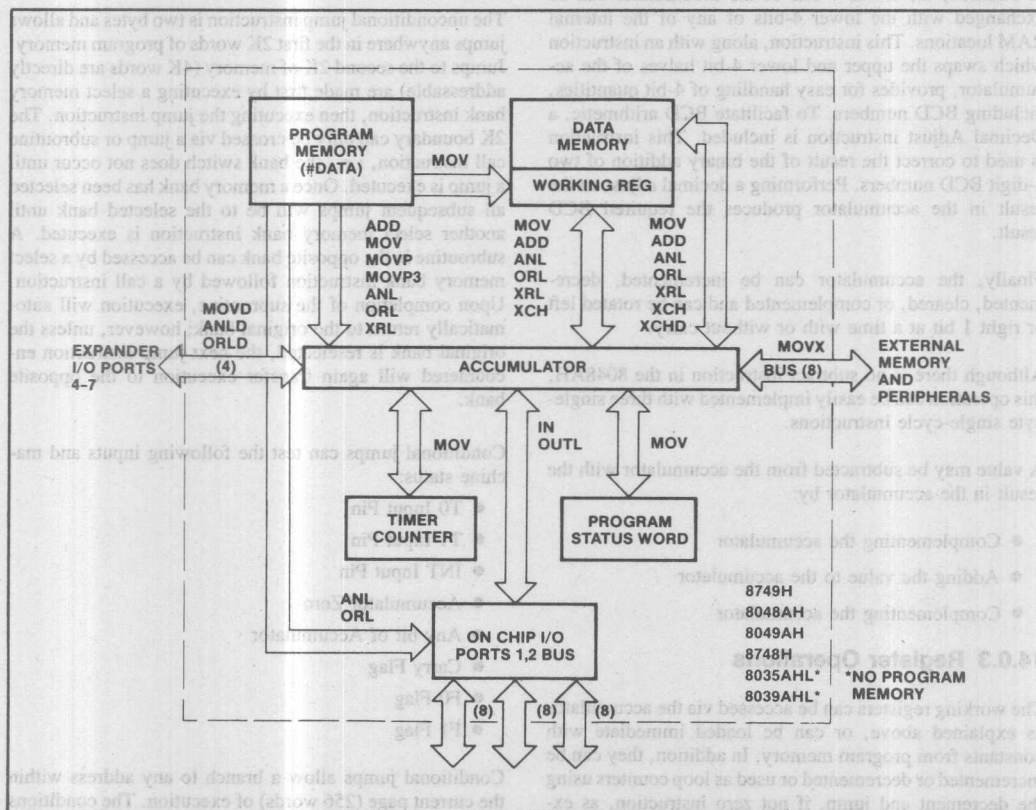


Figure 14-1. Data Transfer Instructions

board timer counter or the accumulator and the Program Status word (PSW). Writing to the PSW alters machine status accordingly and provides a means of restoring status after an interrupt or of altering the stack pointer if necessary.

#### 14.0.2 Accumulator Operations

Immediate data, data memory, or the working registers can be added with or without carry to the accumulator. These sources can also be ANDed, ORed, or Exclusive ORed to the accumulator. Data may be moved to or from the accumulator and working registers or data memory. The two values can also be exchanged in a single operation.

In addition, the lower 4 bits of the accumulator can be exchanged with the lower 4-bits of any of the internal RAM locations. This instruction, along with an instruction which swaps the upper and lower 4-bit halves of the accumulator, provides for easy handling of 4-bit quantities, including BCD numbers. To facilitate BCD arithmetic, a Decimal Adjust instruction is included. This instruction is used to correct the result of the binary addition of two 2-digit BCD numbers. Performing a decimal adjust on the result in the accumulator produces the required BCD result.

Finally, the accumulator can be incremented, decremented, cleared, or complemented and can be rotated left or right 1 bit at a time with or without carry.

Although there is no subtract instruction in the 8048AH, this operation can be easily implemented with three single-byte single-cycle instructions.

A value may be subtracted from the accumulator with the result in the accumulator by:

- Complementing the accumulator
- Adding the value to the accumulator
- Complementing the accumulator

#### 14.0.3 Register Operations

The working registers can be accessed via the accumulator as explained above, or can be loaded immediate with constants from program memory. In addition, they can be incremented or decremented or used as loop counters using the decrement and jump, if not zero instruction, as explained under branch instructions.

All Data Memory including working registers can be accessed with indirect instructions via R0 and R1 and can be incremented.

#### 14.0.4 Flags

There are four user-accessible flags in the 8048AH: Carry, Auxiliary Carry, F0 and F1. Carry indicates overflow of the accumulator, and Auxiliary Carry is used to indicate overflow between BCD digits and is used during decimal-adjust operation. Both Carry and Auxiliary Carry are accessible as part of the program status word and are stored on the stack during subroutines. F0 and F1 are undedicated general-purpose flags to be used as the programmer desires. Both flags can be cleared or complemented and tested by conditional jump instructions. F0 is also accessible via the Program Status word and is stored on the stack with the carry flags.

#### 14.0.5 Branch Instructions

The unconditional jump instruction is two bytes and allows jumps anywhere in the first 2K words of program memory. Jumps to the second 2K of memory (4K words are directly addressable) are made first by executing a select memory bank instruction, then executing the jump instruction. The 2K boundary can only be crossed via a jump or subroutine call instruction, i.e., the bank switch does not occur until a jump is executed. Once a memory bank has been selected all subsequent jumps will be to the selected bank until another select memory bank instruction is executed. A subroutine in the opposite bank can be accessed by a select memory bank instruction followed by a call instruction. Upon completion of the subroutine, execution will automatically return to the original bank; however, unless the original bank is reselected, the next jump instruction encountered will again transfer execution to the opposite bank.

Conditional jumps can test the following inputs and machine status:

- T0 Input Pin
- T1 Input Pin
- INT Input Pin
- Accumulator Zero
- Any bit of Accumulator
- Carry Flag
- F0 Flag
- F1 Flag

Conditional jumps allow a branch to any address within the current page (256 words) of execution. The conditions tested are the instantaneous values at the time the conditional jump is executed. For instance, the jump on accumulator zero instruction tests the accumulator itself, not an intermediate zero flag.

The decrement register and jump if not zero instruction combines a decrement and a branch instruction to create an instruction very useful in implementing a loop counter. This instruction can designate any one of the 8 working registers as a counter and can effect a branch to any address within the current page of execution.

A single-byte indirect jump instruction allows the program to be vectored to any one of several different locations based on the contents of the accumulator. The contents of the accumulator points to a location in program memory which contains the jump address. The 8-bit jump address refers to the current page of execution. This instruction could be used, for instance, to vector to any one of several routines based on an ASCII character which has been loaded in the accumulator. In this way ASCII key inputs can be used to initiate various routines.

#### 14.0.6 Subroutines

Subroutines are entered by executing a call instruction. Calls can be made like unconditional jumps to any address in a 2K word bank, and jumps across the 2K boundary are executed in the same manner. Two separate return instructions determine whether or not status (upper 4-bits of PSW) is restored upon return from the subroutine.

The return and restore status instruction also signals the end of an interrupt service routine if one has been in progress.

#### 14.0.7 Timer Instructions

The 8-bit on board timer/counter can be loaded or read via the accumulator while the counter is stopped or while counting. The counter can be started as a timer with an internal clock source or an event counter or timer with an external clock applied to the T1 input pin. The instruction executed determines which clock source is used. A single instruction stops the counter whether it is operating with an internal or an external clock source. In addition, two instructions allow the timer interrupt to be enabled or disabled.

#### 14.0.8 Control Instructions

Two instructions allow the external interrupt source to be enabled or disabled. Interrupts are initially disabled and are automatically disabled while an interrupt service routine is in progress and re-enabled afterward.

There are four memory bank select instructions, two to designate the active working register bank and two to control program memory banks. The operation of the program memory bank switch is explained in section 13.1.2.

The working register bank switch instructions allow the programmer to immediately substitute a second 8-register working register bank for the one in use. This effectively provides 16 working registers or it can be used as a means of quickly saving the contents of the registers in response to an interrupt. The user has the option to switch or not to switch banks on interrupt. However, if the banks are switched, the original bank will be automatically restored upon execution of a return and restore status instruction at the end of the interrupt service routine.

A special instruction enables an internal clock, which is the XTAL frequency divided by three to be output on pin T0. This clock can be used as a general-purpose clock in the user's system. This instruction should be used only to initialize the system since the clock output can be disabled only by application of system reset.

#### 14.0.9. Input/Output Instructions

Ports 1 and 2 are 8-bit static I/O ports which can be loaded to and from the accumulator. Outputs are statically latched but inputs are not latched and must be read while inputs are present. In addition, immediate data from program memory can be ANDed or ORed directly to Port 1 and Port 2 with the result remaining on the port. This allows "masks" stored in program memory to selectively set or reset individual bits of the I/O ports. Ports 1 and 2 are configured to allow input on a given pin by first writing a "1" out to the pin.

An 8-bit port called BUS can also be accessed via the accumulator and can have statically latched outputs as well. It too can have immediate data ANDed or ORed directly to its outputs, however, unlike ports 1 and 2, all eight lines of BUS must be treated as either input or output at any one time. In addition to being a static port, BUS can be used as a true synchronous bi-directional port using the Move External instructions used to access external data memory. When these instructions are executed, a corresponding READ or WRITE pulse is generated and data is valid only at that time. When data is not being transferred, BUS is in a high impedance state. Note that the OUTL, ANL, and the ORL instructions for the BUS are for use with internal program memory only.

The basic three on-board I/O ports can be expanded via a 4-bit expander bus using half of port 2. I/O expander devices on this bus consist of four 4-bit ports which are addressed as ports 4 through 7. These ports have their own AND and OR instructions like the on-board ports as well as move instructions to transfer data in or out. The expander AND and OR instructions, however, combine the contents of accumulator with the selected port rather than immediate data as is done with the on-board ports.



I/O devices can also be added externally using the BUS port as the expansion bus. In this case the I/O ports become "memory mapped", i.e., they are addressed in the same way as external data memory and exist in the external data memory address space addressed by pointer register R0 or R1.

## 14.1 INSTRUCTION SET DESCRIPTION

The following pages describe the MCS®-48 instruction set in detail. The instruction set is first summarized with instructions grouped functionally. This summary page is followed by a detailed description listed alphabetically by mnemonic opcode.

The alphabetical listing includes the following information:

- Mnemonic
- Machine Code
- Verbal Description
- Symbolic Description
- Assembly Language Example

The machine code is represented with the most significant bit (7) to the left and two byte instructions are represented with the first byte on the left. The assembly language examples are formulated as follows:

Arbitrary

Label: Mnemonic, Operand;

Descriptive Comment

Ports 1 and 2 are 8-bit static I/O ports which can be loaded to and from the accumulator. Outputs are statically latched but inputs are not latched and must be read while inputs are present. In addition, immediate data from program memory can be ANDed or ORed directly to Port 1 and Port 2 with the result remaining on the port. This allows "mask" stored in program memory to selectively set or reset individual bits of the I/O ports. Ports 1 and 2 are configured to allow input on a given pin by first writing a "1" out to the pin.

An 8-bit port called BUS can also be accessed via the accumulator and can have statically latched outputs as well. It too can have immediate data ANDed or ORed directly to its outputs, however, unlike ports 1 and 2, all eight lines of BUS must be treated as either input or output at any one time. In addition to being a static port, BUS can be used as a true synchronous bi-directional port using the Move External Instructions used to access external data memory. When these instructions are executed, a corresponding READ or WRITE pulse is generated and data is valid only at that time. When data is not being transferred, BUS is in a high impedance state. Note that the OUT, ANL, and the ORL instructions for the BUS are for use with internal program memory only.

The basic three on-board I/O ports can be expanded via a 4-bit expander bus using half of port 2. I/O expander devices on this bus consist of four 4-bit ports which are addressed as ports 4 through 7. These ports have their own AND and OR instructions like the on-board ports as well as move instructions to transfer data in or out. The expander AND and OR instructions, however, combine the contents of accumulator with the selected port rather than immediate data as is done with the on-board ports.

Subroutines are entered by executing a call instruction. Calls can be made like unconditional jumps to any address in a 2K word bank, and jumps across the 2K boundary are executed in the same manner. Two separate return instructions determine whether or not status (upper 4-bits of PSW) is removed upon return from the subroutine.

The return and restore status instruction also signals the end of an interrupt service routine if one has been in progress.

The 8-bit on-board timer/counter can be loaded or read via the accumulator while the counter is stopped or while counting. The counter can be started as a timer with an internal clock source or an event counter or timer with an external clock applied to the TI input pin. The instruction executed determines which clock source is used. A single instruction stops the counter whether it is operating with an internal or an external clock source. In addition, two instructions allow the timer interrupt to be enabled or disabled.

Two instructions allow the external interrupt source to be enabled or disabled. Interrupts are initially disabled and are automatically disabled while an interrupt service routine is in progress and re-enabled afterward.

There are four memory bank select instructions, two to designate the active working register bank and two to control program memory banks. The operation of the program memory bank switch is explained in section 13.1.2.

# MCS®-48 INSTRUCTION SET

8048AH/8748H/8049AH/8050AH/8749H

## Instruction Set Summary

Mnemonic	Description	Bytes	Cycle
<b>Accumulator</b>			
ADD A, R	Add register to A	1	1
ADD A, @R	Add data memory to A	1	1
ADD A, # data	Add immediate to A	2	2
ADDC A, R	Add register with carry	1	1
ADDC A, @R	Add data memory with carry	1	1
ADDC A, # data	Add immediate with carry	2	2
ANL A, R	And register to A	1	1
ANL A, @R	And data memory to A	1	1
ANL A, # data	And immediate to A	2	2
ORL A, R	Or register to A	1	1
ORL A, @R	Or data memory to A	1	1
ORL A, # data	Or immediate to A	2	2
XRL A, R	Exclusive Or register to A	1	1
XRL A, @R	Exclusive or data memory to A	1	1
XRL A, # data	Exclusive or immediate to A	2	2
INC A	Increment A	1	1
DEC A	Decrement A	1	1
CLR A	Clear A	1	1
CPL A	Complement A	1	1
DA A	Decimal adjust A	1	1
SWAP A	Swap nibbles of A	1	1
RL A	Rotate A left	1	1
RLC A	Rotate A left through carry	1	1
RR A	Rotate A right	1	1
RRC A	Rotate A right through carry	1	1
<b>Input/Output</b>			
IN A, P	Input port to A	1	2
OUTL P, A	Output A to port	1	2
ANL P, # data	And immediate to port	2	2
ORL P, # data	Or immediate to port	2	2
*INS A, BUS	Input BUS to A	1	2
*OUTL BUS, A	Output A to BUS	1	2
*ANL BUS, # data	And immediate to BUS	2	2
*ORL BUS, # data	Or immediate to BUS	2	2
MOVD A, P	Input Expander port to A	1	2
MOVD P, A	Output A to Expander port	1	2
ANLD P, A	And A to Expander port	1	2
ORLD P, A	Or A to Expander port	1	2

Mnemonic	Description	Bytes	Cycles
<b>Registers</b>			
INC R	Increment register	1	1
INC @R	Increment data memory	1	1
DEC R	Decrement register	1	1
<b>Branch</b>			
JMP addr	Jump unconditional	2	2
JMPP @A	Jump indirect	1	2
DJNZ R, addr	Decrement register and jump	2	2
JC addr	Jump on carry = 1	2	2
JNC addr	Jump on carry = 0	2	2
JZ addr	Jump on A Zero	2	2
JNZ addr	Jump on A not Zero	2	2
JT0 addr	Jump on T0 = 1	2	2
JNT0 addr	Jump on T0 = 0	2	2
JT1 addr	Jump on T1 = 1	2	2
JNT1 addr	Jump on T1 = 0	2	2
JF0 addr	Jump on F0 = 1	2	2
JF1 addr	Jump on F1 = 1	2	2
JTF addr	Jump on timer flag = 1	2	2
JNI addr	Jump on INT = 0	2	2
JBb addr	Jump on Accumulator Bit	2	2
<b>Subroutine</b>			
CALL addr	Jump to subroutine	2	2
RET	Return	1	2
RETR	Return and restore status	1	2
<b>Flags</b>			
CLR C	Clear Carry	1	1
CPL C	Complement Carry	1	1
CLR F0	Clear Flag 0	1	1
CPL F0	Complement Flag 0	1	1
CLR F1	Clear Flag 1	1	1
CPL F1	Complement Flag 1	1	1
<b>Data Moves</b>			
MOV A, R	Move register to A	1	1
MOV A, @R	Move data memory to A	1	1
MOV A, # data	Move immediate to A	2	2
MOV R, A	Move A to register	1	1
MOV @R, A	Move A to data memory	1	1
MOV R, # data	Move immediate to register	2	2
MOV @R, # data	Move immediate to data memory	2	2
MOV A, PSW	Move PSW to A	1	1
MOV PSW, A	Move A to PSW	1	1

Mnemonics copyright Intel Corporation 1983.

\*For use with internal memory only.

# MCS®-48 INSTRUCTION SET

## 8048AH/8748H/8049AH/8050AH/8749H Instruction Set Summary (Con't)

Mnemonic	Description	Bytes	Cycle
<b>Data Moves (Cont'd)</b>			
XCH A, R	Exchange A and register	1	1
XCH A, @R	Exchange A and data memory	1	1
XCHD A, @R	Exchange nibble of A and register	1	1
MOVX A, @R	Move external data memory to A	1	2
MOVX @R, A	Move A to external data memory	1	2
MOVP A, @A	Move to A from current page	1	2
MOVP3 A, @A	Move to A from Page 3	1	2
<b>Timer/Counter</b>			
MOV A, T	Read Timer/Counter	1	1
MOV T, A	Load Timer/Counter	1	1
STRT T	Start Timer	1	1
STRT CNT	Start Counter	1	1
STOP TCNT	Stop Timer/Counter	1	1
EN TCNTI	Enable Timer/Counter Interrupt	1	1
DIS TCNTI	Disable Timer/Counter Interrupt	1	1

Mnemonic	Description	Bytes	Cycle
<b>Control</b>			
EN I	Enable external Interrupt	1	1
DIS I	Disable external Interrupt	1	1
SEL RB0	Select register bank 0	1	1
SEL RB1	Select register bank 1	1	1
SEL MB0	Select memory bank 0	1	1
SEL MB1	Select memory bank 1	1	1
ENT0 CLK	Enable clock output on T0	1	1
NOP	No Operation	1	1

Mnemonics copyright Intel Corporation 1983.

IN A, P	Input port to A	1	2
OUT A, P	Output A to port	1	2
IN A, P, & data	And immediate to port	2	2
OUT A, P, & data	Or immediate to port	2	2
IN A, BUS	Input BUS to A	1	2
OUT A, BUS	Output A to BUS	1	2
IN A, BUS, & data	And immediate to BUS	2	2
OUT A, BUS, & data	Or immediate to BUS	2	2
MOV A, P	Input Expander port to A	1	2
MOV P, A	Output A to Expander port	1	2
AND P, A	And A to Expander port	1	2
OR P, A	Or A to Expander port	1	2

Mnemonics copyright Intel Corporation 1983.  
For use with internal memory only.

## MCS®-48 INSTRUCTION SET

### Symbols and Abbreviations Used

A	Accumulator
AC	Auxiliary Carry
addr	12-Bit Program Memory Address
Bb	Bit Designator (b = 0-7)
BS	Bank Switch
BUS	BUS Port
C	Carry
CLK	Clock
CNT	Event Counter
CRR	Conversion Result Register
D	Mnemonic for 4-Bit Digit (Nibble)
data	8-Bit Number or Expression
DBF	Memory Bank Flip-Flop
F0, F1	Flag 0, Flag 1
I	Interrupt
P	Mnemonic for "in-page" Operation
PC	Program Counter
Pp	Port Designator (p = 1, 2 or 4-7)
PSW	Program Status Word
Ri	Data memory Pointer (i = 0, or 1)
Rr	Register Designator (r = 0-7)
SP	Stack Pointer
T	Timer
TF	Timer Flag
T0, T1	Test 0, Test 1
X	Mnemonic for External RAM
#	Immediate Data Prefix
@	Indirect Address Prefix
\$	Current Value of Program Counter
(X)	Contents of X
((X))	Contents of Location Addressed by X
←	Is Replaced by

Mnemonics copyright Intel Corporation 1983.



**ADD A,R<sub>r</sub> Add Register Contents to Accumulator****Encoding:**

0	1	1	0	1	r	r	r
---	---	---	---	---	---	---	---

 68H-6FH**Description:** The contents of register 'r' are added to the accumulator. Carry is affected.**Operation:**  $(A) \leftarrow (A) + (Rr)$  r = 0-7**Example:** ADDREG: ADD A,R6 ;ADD REG 6 CONTENTS  
;TO ACC**ADD A,@R<sub>i</sub> Add Data Memory Contents to Accumulator****Encoding:**

0	1	1	0	0	0	0	i
---	---	---	---	---	---	---	---

 60H-61H**Description:** The contents of the resident data memory location addressed by register 'i' bits 0-5\*\* are added to the accumulator. Carry is affected.**Operation:**  $(A) \leftarrow (A) + ((Ri))$  i = 0-1**Example:** ADDM: MOV R0, #01FH ;MOVE '1F' HEX TO REG 0  
ADD A, @R0 ;ADD VALUE OF LOCATION  
;31 TO ACC**ADD A,#data Add Immediate Data to Accumulator****Encoding:**

0	0	0	0	0	0	1	1
---	---	---	---	---	---	---	---

d <sub>7</sub>	d <sub>6</sub>	d <sub>5</sub>	d <sub>4</sub>	d <sub>3</sub>	d <sub>2</sub>	d <sub>1</sub>	d <sub>0</sub>
----------------	----------------	----------------	----------------	----------------	----------------	----------------	----------------

 03H**Description:** This is a 2-cycle instruction. The specified data is added to the accumulator. Carry is affected.**Operation:**  $(A) \leftarrow (A) + \text{data}$ **Example:** ADDID: ADD A,#ADDER: ;ADD VALUE OF SYMBOL  
;ADDER TO ACC**ADDC A,R<sub>r</sub> Add Carry and Register Contents to Accumulator****Encoding:**

0	1	1	1	1	r	r	r
---	---	---	---	---	---	---	---

 78H-7FH**Description:** The content of the carry bit is added to accumulator location 0 and the carry bit cleared. The contents of register 'r' are then added to the accumulator. Carry is affected.**Operation:**  $(A) \leftarrow (A) + (Rr) + (C)$  r = 0-7**Example:** ADDRGC: ADDC A,R4 ;ADD CARRY AND REG 4  
;CONTENTS TO ACC

\*\* 0-5 in 8048AH/8748H

0-6 in 8049AH/8749H

0-7 in 8050AH

### ADDC A,@R<sub>i</sub> Add Carry and Data Memory Contents to Accumulator

Encoding: 

0	1	1	1	0	0	0	i
---	---	---	---	---	---	---	---

 70H-71H

**Description:** The content of the carry bit is added to accumulator location 0 and the carry bit cleared. Then the contents of the resident data memory location addressed by register 'i' bits 0-5\*\* are added to the accumulator. Carry is affected.

**Operation:**  $(A) \leftarrow (A) + ((R_i)) + (C)$   $i = 0-1$

**Example:** ADDMC: MOV R1,#40 ;MOVE '40' DEC TO REG 1  
ADDC A,@R1 ;ADD CARRY AND LOCATION 40  
;CONTENTS TO ACC

### ADDC A,@data Add Carry and Immediate Data to Accumulator

Encoding: 

0	0	0	1
---	---	---	---

d <sub>7</sub>	d <sub>6</sub>	d <sub>5</sub>	d <sub>4</sub>	d <sub>3</sub>	d <sub>2</sub>	d <sub>1</sub>	d <sub>0</sub>
----------------	----------------	----------------	----------------	----------------	----------------	----------------	----------------

 13H

**Description:** This is a 2-cycle instruction. The content of the carry bit is added to accumulator location 0 and the carry bit cleared. Then the specified data is added to the accumulator. Carry is affected.

**Operation:**  $(A) \leftarrow (A) + \text{data} + (C)$

**Example:** ADDC A,#225 ;ADD CARRY AND '225' DEC  
;TO ACC

### ANL A,R<sub>r</sub> Logical AND Accumulator with Register Mask

Encoding: 

0	1	0	1	1	r	r	r
---	---	---	---	---	---	---	---

 58H-5FH

**Description:** Data in the accumulator is logically ANDed with the mask contained in working register 'r'.

**Operation:**  $(A) \leftarrow (A) \text{ AND } (R_r)$   $r = 0-7$

**Example:** ANDREG: ANL A,R3 ;'AND' ACC CONTENTS WITH MASK  
;IN REG 3

### ANL A,@R<sub>i</sub> Logical AND Accumulator with memory Mask

Encoding: 

0	1	0	1	0	0	0	i
---	---	---	---	---	---	---	---

 50H-51H

**Description:** Data in the accumulator is logically ANDed with the mask contained in the data memory location referenced by register 'i' bits 0-5\*\*.

**Operation:**  $(A) \leftarrow (A) \text{ AND } ((R_i))$   $i = 0-1$

**Example:** ANDDM: MOV R0,#03FH ;MOVE '3F' HEX TO REG 0  
ANL A,@R0 ;'AND' ACC CONTENTS WITH  
;MASK IN LOCATION 63

\*\* 0-5 in 8048AH/8748H  
0-6 in 8049AH/8749H  
0-7 in 8050AH

**ANL A,#data Logical AND Accumulator with Immediate Mask**

**Encoding:**

0	1	0	1
---	---	---	---

0	0	1	1
---	---	---	---

d <sub>7</sub>	d <sub>6</sub>	d <sub>5</sub>	d <sub>4</sub>
----------------	----------------	----------------	----------------

d <sub>3</sub>	d <sub>2</sub>	d <sub>1</sub>	d <sub>0</sub>
----------------	----------------	----------------	----------------

 53H

**Description:** This is a 2-cycle instruction. Data in the accumulator is logically ANDed with an immediately-specified mask.

**Operation:** (A) ← (A) AND data

**Examples:** ANDID: ANL A,#0AFH

;AND' ACC CONTENTS

;WITH MASK 10101111

ANL A,#3 + X/Y

;AND' ACC CONTENTS

;WITH VALUE OF EXP

;3 + XY/Y'

**ANL BUS,#data\* Logical AND BUS with Immediate Mask**

**Encoding:**

1	0	0	1
---	---	---	---

1	0	0	0
---	---	---	---

d <sub>7</sub>	d <sub>6</sub>	d <sub>5</sub>	d <sub>4</sub>
----------------	----------------	----------------	----------------

d <sub>3</sub>	d <sub>2</sub>	d <sub>1</sub>	d <sub>0</sub>
----------------	----------------	----------------	----------------

 98H

**Description:** This is a 2-cycle instruction. Data on the BUS port is logically ANDed with an immediately-specified mask. This instruction assumes prior specification of an 'OUTL BUS, A' instruction.

**Operation:** (BUS) ← (BUS) AND data

**Example:** ANDBUS: ANL BUS,#MASK

;AND' BUS CONTENTS

;WITH MASK EQUAL VALUE

;OF SYMBOL 'MASK'

**ANL Pp,#data Logical AND Port 1-2 with Immediate Mask**

**Encoding:**

1	0	0	1
---	---	---	---

1	0	p	p
---	---	---	---

d <sub>7</sub>	d <sub>6</sub>	d <sub>5</sub>	d <sub>4</sub>
----------------	----------------	----------------	----------------

d <sub>3</sub>	d <sub>2</sub>	d <sub>1</sub>	d <sub>0</sub>
----------------	----------------	----------------	----------------

 99H-9AH

**Description:** This is a 2-cycle instruction. Data on port 'p' is logically ANDed with an immediately-specified mask.

**Operation:** (Pp) ← (Pp) AND DATA

p = 1-2

**Example:** ANDP2: ANL P2,#0F0H

;AND' PORT 2 CONTENTS

;WITH MASK 'F0' HEX

;(CLEAR P20-23)

\* For use with internal program memory ONLY.

**ANLD Pp,A Logical AND Port 4-7 with Accumulator Mask**

**Encoding:**

1	0	0	1	1	1	p	p
---	---	---	---	---	---	---	---

 9CH-9FH

**Description:** This is a 2-cycle instruction. Data on port 'p' is logically ANDed with the digit mask contained in accumulator bits 0-3.

**Operation:**  $(Pp) \leftarrow (Pp) \text{ AND } (A0-3)$  p = 4-7

Note: The mapping of port 'p' to opcode bits 0-1 is as follows:

10	Port
00	4
01	5
10	6
11	7

**Example:** ANDP4: ANLD P4,A ;'AND' PORT 4 CONTENTS  
;WITH ACC BITS 0-3

**CALL address Subroutine Call**

**Encoding:**

a <sub>10</sub>	a <sub>9</sub>	a <sub>8</sub>	1	0	1	0	0
-----------------	----------------	----------------	---	---	---	---	---

a <sub>7</sub>	a <sub>6</sub>	a <sub>5</sub>	a <sub>4</sub>	a <sub>3</sub>	a <sub>2</sub>	a <sub>1</sub>	a <sub>0</sub>
----------------	----------------	----------------	----------------	----------------	----------------	----------------	----------------

Page	Hex Op Code
0	14
1	34
2	54
3	74
4	94
5	B4
6	D4
7	F4

**Description:** This is a 2-cycle instruction. The program counter and PSW bits 4-7 are saved in the stack. The stack pointer (PSW bits 0-2) is updated. Program control is then passed to the location specified by 'address'. PC bit 11 is determined by the most recent SEL MB instruction.

A CALL cannot begin in locations 2046-2047 or 4094-4095. Execution continues at the instruction following the CALL upon return from the subroutine.

**Operation:**  $((SP)) \leftarrow (PC), (PSW_{4-7})$   
 $(SP) \leftarrow (SP) + 1$   
 $(PC_{8-10}) \leftarrow (addr_{8-10})$   
 $(PC_{0-7}) \leftarrow (addr_{0-7})$   
 $(PC_{11}) \leftarrow DBF$



# MCS®-48 INSTRUCTION SET

**Example:** Add three groups of two numbers. Put subtotals in locations 50, 51 and total in location 52.

```

MOV R0,#50      ;MOVE '50' DEC TO ADDRESS
                  ;REG 0
BEGADD: MOV A,R1  ;MOVE CONTENTS OF REG 1
                  ;TO ACC
          ADD A,R2  ;ADD REG 2 TO ACC
          CALL SUBTOT ;CALL SUBROUTINE 'SUBTOT'
          ADDC A,R3  ;ADD REG 3 TO ACC
          ADDC A,R4  ;ADD REG 4 TO ACC
          CALL SUBTOT ;CALL SUBROUTINE 'SUBTOT'
          ADDC A,R5  ;ADD REG 5 TO ACC
          ADDC A,R6  ;ADD REG 6 TO ACC
          CALL SUBTOT ;CALL SUBROUTINE 'SUBTOT'
SUBTOT: MOV @R0,A  ;MOVE CONTENTS OF ACC TO
                  ;LOCATION ADDRESSED BY
                  ;REG 0
          INC R0    ;INCREMENT REG 0
          RET       ;RETURN TO MAIN PROGRAM
    
```

## CLR A Clear Accumulator

**Encoding:** 0 0 1 0 0 1 1 1 27H

**Description:** The contents of the accumulator are cleared to zero.

**Operation:**  $A \leftarrow 0$

## CLR C Clear Carry Bit

**Encoding:** 1 0 0 1 0 1 1 1 97H

**Description:** During normal program execution, the carry bit can be set to one by the ADD, ADDC, RLC, CPL C, RRC, and DAA instructions. This instruction resets the carry bit to zero.

**Operation:**  $C \leftarrow 0$

## CLR F1 Clear Flag 1

**Encoding:** 1 0 1 0 0 1 0 1 A5H

**Description:** Flag 1 is cleared to zero.

**Operation:**  $(F1) \leftarrow 0$

## CLR F0 Clear Flag 0

Encoding: 

1	0	0	0
0	1	0	1

 85H

Description: Flag 0 is cleared to zero.

Operation: (F0) ← 0

## CPL A Complement Accumulator

Encoding: 

0	0	1	1
0	1	1	1

 37H

Description: The contents of the accumulator are complemented. This is strictly a one's complement. Each one is changed to zero and vice-versa.

Operation: (A) ← NOT (A)

Example: Assume accumulator contains 01101010.

CPLA: CPL A

;ACC CONTENTS ARE COMPLEMENTED TO 10010101

## CPL C Complement Carry Bit

Encoding: 

1	0	1	0
0	1	1	1

 A7H

Description: The setting of the carry bit is complemented; one is changed to zero, and zero is changed to one.

Operation: (C) ← NOT (C)

Example: Set C to one; current setting is unknown.

CTO1: CLR C

;C IS CLEARED TO ZERO

CPL C

;C IS SET TO ONE

## CPL F0 Complement Flag 0

Encoding: 

1	0	0	1
0	1	0	1

 95H

Description: The setting of flag 0 is complemented; one is changed to zero, and zero is changed to one.

Operation: F0 ← NOT (F0)

## CPL F1 Complement Flag 1

Encoding: 

1	0	1	1
0	1	0	1

 B5H

Description: The setting of flag 1 is complemented; one is changed to zero, and zero is changed to one.

Operation: (F1) ← NOT (F1)

**DA A Decimal Adjust Accumulator****Encoding:**

0	1	0	1
---	---	---	---

0	1	1	1
---	---	---	---

 57H

**Description:** The 8-bit accumulator value is adjusted to form two 4-bit Binary Coded Decimal (BCD) digits following the binary addition of BCD numbers. The carry bit C is affected. If the contents of bits 0-3 are greater than nine, or if AC is one, the accumulator is incremented by six.

The four high-order bits are then checked. If bits 4-7 exceed nine, or if C is one, these bits are increased by six. If an overflow occurs, C is set to one.

**Example:** Assume accumulator contains 10011011.  
 DA A ;ACC Adjusted to 00000001  
 ;WITH C SET

C	AC	7	4	3	0	
0	0	1	0	0	1	1 0 1 1
					0	0 0 0 0 1 1 0
0	1	1	0	1	0	0 0 0 1
					0	1 1 0
1	0	0	0	0	0	0 0 1

ADD SIX TO BITS 0-7

ADD SIX TO BITS 4-7

OVERFLOW TO C

**DEC A Decrement Accumulator****Encoding:**

0	0	0	0
---	---	---	---

0	1	1	1
---	---	---	---

 07H

**Description:** The contents of the accumulator are decremented by one. The carry flag is not affected.

**Operation:** (A) ← (A) - 1

**Example:** Decrement contents of external data memory location 63.

MOV R0,#3FH ;MOVE '3F' HEX TO REG 0  
 MOVX A, @R0 ;MOVE CONTENTS OF  
 ;LOCATION 63 TO ACC

DEC A ;DECREMENT ACC  
 MOVX @R0,A ;MOVE CONTENTS OF ACC TO  
 ;LOCATION 63 IN EXPANDED  
 ;MEMORY

**DEC Rr Decrement Register****Encoding:**

1	1	0	0
---	---	---	---

1	r	r	r
---	---	---	---

 C8H-CFH

**Description:** The contents of working register 'r' are decremented by one.

**Operation:** (Rr) ← (Rr) - 1 r = 0-7

**Example:** DEC R1 ;DECREMENT CONTENTS OF REG 1

**DIS I External Interrupt****Encoding:** 0 0 0 1 0 1 0 1 15H**Description:** External interrupts are disabled. A low signal on the interrupt input pin has no effect.**DIS TCNTI Disable Timer/Counter Interrupt****Encoding:** 0 0 1 1 0 1 0 1 35H**Description:** Timer/counter interrupts are disabled. Any pending timer interrupt request is cleared. The interrupt sequence is not initiated by an overflow, but the timer flag is set and time accumulation continues.**DJNZ R<sub>r</sub>, address Decrement Register and Test****Encoding:** 1 1 1 0 1 r r r a<sub>7</sub> a<sub>6</sub> a<sub>5</sub> a<sub>4</sub> a<sub>3</sub> a<sub>2</sub> a<sub>1</sub> a<sub>0</sub> E8H-EFH**Description:** This is a 2-cycle instruction. Register 'r' is decremented, then tested for zero. If the register contains all zeros, program control falls through to the next instruction. If the register contents are not zero, control jumps to the specified 'address'.

The address in this case must evaluate to 8-bits, that is, the jump must be to a location within the current 256-location page.

**Example:** (Rr) ← (Rr) - 1 r = 0-7

If Rr not 0

(PC<sub>0-7</sub>) ← addr**Note:** A 12-bit address specification does not cause an error if the DJNZ instruction and the jump target are on the same page. If the DJNZ instruction begins in location 255 of a page, it must jump to a target address on the following page.**Example:** Increment values in data memory locations 50-54.

MOV R0,#50 ;MOVE '50' DEC TO ADDRESS

;REG 0

MOV R3,#5 ;MOVE '5' DEC TO COUNTER

;REG 3

INCRT: INC @R0 ;INCREMENT CONTENTS OF

;LOCATION ADDRESSED BY

;REG 0

INC R0 ;INCREMENT ADDRESS IN REG 0

DJNZ R3, INCRT ;DECREMENT REG 3 — JUMP TO

;'INCRT' IF REG 3 NONZERO

NEXT — ;'NEXT' ROUTINE EXECUTED

;IF R3 IS ZERO



## EN I Enable External Interrupt

Encoding: 

0	0	0	0
0	1	0	1

 05H

**Description:** External interrupts are enabled. A low signal on the interrupt input pin initiates the interrupt sequence.

## EN TCNTI Enable Timer/Counter Interrupt

Encoding: 

0	0	1	0
0	1	0	1

 25H

**Description:** Timer/counter interrupts are enabled. An overflow of the timer/counter initiates the interrupt sequence.

## ENT0 CLK Enable Clock Output

Encoding: 

0	1	1	1
0	1	0	1

 75H

**Description:** The test 0 pin is enabled to act as the clock output. This function is disabled by a system reset.

**Example:** EMTST0: ENT0 CLK ;ENABLE T0 AS CLOCK OUTPUT

## IN A,Pp Input Port or Data to Accumulator

Encoding: 

0	0	0	0
1	0	p	p

 09H-0AH

**Description:** This is a 2-cycle instruction. Data present on port 'p' is transferred (read) to the accumulator.

**Operation:** (A) ← (Pp) p = 1-2

INP12: IN A,P1 ;INPUT PORT 1 CONTENTS TO ACC  
MOV R6,A ;MOVE ACC CONTENTS TO REG 6  
IN A,P2 ;INPUT PORT 2 CONTENTS TO ACC  
MOV R7,A ;MOVE ACC CONTENTS TO REG 7

## INC A Increment Accumulator

Encoding: 

0	0	0	1
0	1	1	1

 17H

**Description:** The contents of the accumulator are incremented by one. Carry is not affected.

**Operation:** (A) ← (A) + 1

**Example:** Increment contents of location 100 in external data memory.

```
INCA: MOV R0,#100      ;MOVE '100' DEC TO ADDRESS REG 0
      MOVX A,@R0       ;MOVE CONTENTS OF LOCATION
                        ;100 TO ACC
      INC A            ;INCREMENT A
      MOVX @R0,A       ;MOVE ACC CONTENTS TO
                        ;LOCATION 101
```

### INC R<sub>r</sub> Increment Register

**Encoding:**

0	0	0	1	1	r	r	r
---	---	---	---	---	---	---	---

 18H-1FH

**Description:** The contents of working register 'r' are incremented by one.

**Operation:**  $(R_r) \leftarrow (R_r) + 1$  r = 0-7

**Example:** INCR0: INC R0 ;INCREMENT CONTENTS OF REG 0

### INC @R<sub>i</sub> Increment Data Memory Location

**Encoding:**

0	0	0	1	0	0	0	i
---	---	---	---	---	---	---	---

 10H-11H

**Description:** The contents of the resident data memory location addressed by register 'i' bits 0-5\*\* are incremented by one.

**Operation:**  $((R_i)) \leftarrow ((R_i)) + 1$  i = 0-1

**Example:** INCDM: MOV R1,#03FH ;MOVE ONES TO REG 1  
INC @R1 ;INCREMENT LOCATION 63

### INS A,BUS\* Strobed Input of BUS Data to Accumulator

**Encoding:**

0	0	0	0	1	0	0	0
---	---	---	---	---	---	---	---

 08H

**Description:** This is a 2-cycle instruction. Data present on the BUS port is transferred (read) to the accumulator when the RD pulse is dropped. (Refer to section on programming memory expansion for details.)

**Operation:**  $(A) \leftarrow (BUS)$

**Example:** INPBUS: INS A,BUS ;INPUT BUS CONTENTS TO ACC

\* For use with internal program memory ONLY.

\*\* 0-5 in 8048AH/8748H

0-6 in 8049AH/8749H

0-7 in 8050AH

**JBb address Jump If Accumulator Bit Is Set**

Encoding: 

b <sub>2</sub>	b <sub>1</sub>	b <sub>0</sub>	1	0	0	1	0
----------------	----------------	----------------	---	---	---	---	---

a <sub>7</sub>	a <sub>6</sub>	a <sub>5</sub>	a <sub>4</sub>	a <sub>3</sub>	a <sub>2</sub>	a <sub>1</sub>	a <sub>0</sub>
----------------	----------------	----------------	----------------	----------------	----------------	----------------	----------------

Accumulator Bit	Hex Op Code
0	12
1	32
2	52
3	72
4	92
5	B2
6	D2
7	F2

**Description:** This is a 2-cycle instruction. Control passes to the specified address if accumulator bit 'b' is set to one.

**Operation:** b = 0-7

(PC<sub>0-7</sub>) ← addr

(PC) = (PC) + 2

**Example:** JB4IS1: JB4 NEXT

If Bb = 1

If Bb = 0

;JUMP TO 'NEXT' ROUTINE

;IF ACC BIT 4 = 1

**JC address Jump If Carry Is Set**

Encoding: 

1	1	1	1
---	---	---	---

0	1	1	0
---	---	---	---

a <sub>7</sub>	a <sub>6</sub>	a <sub>5</sub>	a <sub>4</sub>	a <sub>3</sub>	a <sub>2</sub>	a <sub>1</sub>	a <sub>0</sub>
----------------	----------------	----------------	----------------	----------------	----------------	----------------	----------------

 F6H

**Description:** This is a 2-cycle instruction. Control passes to the specified address if the carry bit is set to one.

**Operation:** (PC<sub>0-7</sub>) ← addr

(PC) = (PC) + 2

**Example:** JC1: JC OVFLOW

If C = 1

If C = 0

;JUMP TO 'OVFLOW' ROUTINE

;IF C = 1

**JF0 address Jump If Flag 0 Is Set**

Encoding: 

1	0	1	1
---	---	---	---

0	1	1	0
---	---	---	---

a <sub>7</sub>	a <sub>6</sub>	a <sub>5</sub>	a <sub>4</sub>	a <sub>3</sub>	a <sub>2</sub>	a <sub>1</sub>	a <sub>0</sub>
----------------	----------------	----------------	----------------	----------------	----------------	----------------	----------------

 B6H

**Description:** This is a 2-cycle instruction. Control passes to the specified address if flag 0 is set to one.

**Operation:** (PC<sub>0-7</sub>) ← addr

(PC) = (PC) + 2

**Example:** JF0IS1: JF0 TOTAL

If F0 = 1

If F0 = 0

;JUMP TO 'TOTAL' ROUTINE IF F0 = 1

**JF1 address Jump If Flag 1 Is Set**

**Encoding:**

0	1	1	1
---	---	---	---

0	1	1	0
---	---	---	---

a <sub>7</sub>	a <sub>6</sub>	a <sub>5</sub>	a <sub>4</sub>
----------------	----------------	----------------	----------------

a <sub>3</sub>	a <sub>2</sub>	a <sub>1</sub>	a <sub>0</sub>
----------------	----------------	----------------	----------------

 76H

**Description:** This is a 2-cycle instruction. Control passes to the specified address if flag 1 is set to one.

**Operation:**  $(PC_{0-7}) \leftarrow \text{addr}$  If F1 = 1  
 $(PC) = (PC + 2)$  If F1 = 0

**Example:** JF1S1: JF1 FILBUF ;JUMP TO 'FILBUF'  
 ;ROUTINE IF F1 = 1

**JMP address Direct Jump within 2K Block**

**Encoding:**

a <sub>10</sub>	a <sub>9</sub>	a <sub>8</sub>	0
-----------------	----------------	----------------	---

0	1	0	0
---	---	---	---

a <sub>7</sub>	a <sub>6</sub>	a <sub>5</sub>	a <sub>4</sub>
----------------	----------------	----------------	----------------

a <sub>3</sub>	a <sub>2</sub>	a <sub>1</sub>	a <sub>0</sub>
----------------	----------------	----------------	----------------

Page	Hex Op Code
0	04
1	24
2	44
3	64
4	84
5	A4
6	C4
7	E4

**Description:** This is a 2-cycle instruction. Bits 0-10 of the program counter are replaced with the directly-specified address. The setting of PC bit 11 is determined by the most recent SELECT MB instruction.

**Operation:**  $(PC_{8-10}) \leftarrow \text{addr } 8-10$   
 $(PC_{0-7}) \leftarrow \text{addr } 0-7$   
 $(PC_{11}) \leftarrow \text{DBF}$

**Example:** JMP SUBTOT ;JUMP TO SUBROUTINE 'SUBTOT'  
 JMP \$-6 ;JUMP TO INSTRUCTION SIX  
 ;LOCATIONS BEFORE CURRENT  
 ;LOCATION  
 JMP 2FH ;JUMP TO ADDRESS '2F' HEX

**JMPP @A Indirect Jump within Page**

**Encoding:**

1	0	1	1
---	---	---	---

0	0	1	1
---	---	---	---

 B3H

**Description:** This is a 2-cycle instruction. The contents of the program memory location pointed to by the accumulator are substituted for the 'page' portion of the program counter (PC bits 0-7).



**Operation:**  $(PC_{0-7}) \leftarrow ((A))$

**Example:** Assume accumulator contains 0FH.

JMPPAG: JMPP @A

;JUMP TO ADDRESS STORED IN  
;LOCATION 15 IN CURRENT PAGE

#### JNC address Jump If Carry Is Not Set

**Encoding:**

1	1	1	0	0	1	1	0
---	---	---	---	---	---	---	---

a <sub>7</sub>	a <sub>6</sub>	a <sub>5</sub>	a <sub>4</sub>	a <sub>3</sub>	a <sub>2</sub>	a <sub>1</sub>	a <sub>0</sub>
----------------	----------------	----------------	----------------	----------------	----------------	----------------	----------------

 E6H

**Description:** This is a 2-cycle instruction. Control passes to the specified address if the carry bit is not set, that is, equals zero.

**Operation:**  $(PC_{0-7}) \leftarrow \text{addr}$  If C = 0  
 $(PC) = (PC) + 2$  If C = 1

**Example:** JC0: JNC NOVFO ;JUMP TO 'NOVFO' ROUTINE  
;IF C = 0

#### JNI address Jump If Interrupt Input Is Low

**Encoding:**

1	0	0	0	0	1	1	0
---	---	---	---	---	---	---	---

a <sub>7</sub>	a <sub>6</sub>	a <sub>5</sub>	a <sub>4</sub>	a <sub>3</sub>	a <sub>2</sub>	a <sub>1</sub>	a <sub>0</sub>
----------------	----------------	----------------	----------------	----------------	----------------	----------------	----------------

 86H

**Description:** This is a 2-cycle instruction. Control passes to the specified address if the interrupt input signal is low (= 0), that is, an external interrupt has been signaled. (This signal initiates an interrupt service sequence if the external interrupt is enabled.)

**Operation:**  $(PC_{0-7}) \leftarrow \text{addr}$  If I = 0  
 $(PC) = (PC) + 2$  If I = 1

**Example:** LOC 3: JNI EXTINT ;JUMP TO 'EXTINT' ROUTINE  
;IF I = 0

#### JNT0 address Jump If Test 0 is Low

**Encoding:**

0	0	1	0	0	1	1	0
---	---	---	---	---	---	---	---

a <sub>7</sub>	a <sub>6</sub>	a <sub>5</sub>	a <sub>4</sub>	a <sub>3</sub>	a <sub>2</sub>	a <sub>1</sub>	a <sub>0</sub>
----------------	----------------	----------------	----------------	----------------	----------------	----------------	----------------

 26H

**Description:** This is a 2-cycle instruction. Control passes to the specified address, if the test 0 signal is low.

**Operation:**  $(PC_{0-7}) \leftarrow \text{addr}$  If T0 = 0  
 $(PC) = (PC) + 2$  If T0 = 1

**Example:** JT0LOW: JNT0 60 ;JUMP TO LOCATION 60 DEC  
;IF T0 = 0

# MCS®-48 INSTRUCTION SET

## JNT1 address Jump If Test 1 Is Low

**Encoding:** 0 1 0 0 0 1 1 0 a<sub>7</sub> a<sub>6</sub> a<sub>5</sub> a<sub>4</sub> a<sub>3</sub> a<sub>2</sub> a<sub>1</sub> a<sub>0</sub> 46H

**Description:** This is a 2-cycle instruction. Control passes to the specified address, if the test 1 signal is low.

**Operation:** (PC<sub>0-7</sub>) ← addr If T1 = 0  
(PC) = (PC) + 2 If T1 = 1

## JNZ Address Jump If Accumulator Is Not Zero

**Encoding:** 1 0 0 1 0 1 1 0 a<sub>7</sub> a<sub>6</sub> a<sub>5</sub> a<sub>4</sub> a<sub>3</sub> a<sub>2</sub> a<sub>1</sub> a<sub>0</sub> 96H

**Description:** This is a 2-cycle instruction. Control passes to the specified address if the accumulator contents are nonzero at the time this instruction is executed.

**Operation:** (PC<sub>0-7</sub>) ← addr If A ≠ 0  
(PC) = (PC) + 2 If A = 0

**Example:** JACCN0: JNZ 0ABH ;JUMP TO LOCATION 'AB' HEX  
;IF ACC VALUE IS NONZERO

## JTF address Jump If Timer Flag Is Set

**Encoding:** 0 0 0 1 0 1 1 0 a<sub>7</sub> a<sub>6</sub> a<sub>5</sub> a<sub>4</sub> a<sub>3</sub> a<sub>2</sub> a<sub>1</sub> a<sub>0</sub> 16H

**Description:** This is a 2-cycle instruction. Control passes to the specified address if the timer flag is set to one, that is, the timer/counter register has overflowed. Testing the timer flag resets it to zero. (This overflow initiates an interrupt service sequence if the timer-overflow interrupt is enabled.)

**Operation:** (PC<sub>0-7</sub>) ← addr If TF = 1  
(PC) = (PC) + 2 If TF = 0

**Example:** JTF1: JTF TIMER ;JUMP TO 'TIMER' ROUTINE  
;IF TF = 1

## JT0 address Jump If Test 0 Is High

**Encoding:** 0 0 1 1 0 1 1 0 a<sub>7</sub> a<sub>6</sub> a<sub>5</sub> a<sub>4</sub> a<sub>3</sub> a<sub>2</sub> a<sub>1</sub> a<sub>0</sub> 36H

**Description:** This is a 2-cycle instruction. Control passes to the specified address if the test 0 signal is high (= 1).

**Operation:** (PC<sub>0-7</sub>) ← addr If T0 = 1  
(PC) = (PC) + 2 If T0 = 0

**Example:** JT0HI: JT0 53 ;JUMP TO LOCATION 53 DEC  
;IF T0 = 1

**JT1 address    Jump If Test 1 Is High**

**Encoding:** 0 1 0 1    0 1 1 0    a<sub>7</sub> a<sub>6</sub> a<sub>5</sub> a<sub>4</sub>    a<sub>3</sub> a<sub>2</sub> a<sub>1</sub> a<sub>0</sub>    56H

**Description:** This is a 2-cycle instruction. Control passes to the specified address if the test 1 signal is high (= 1).

**Operation:** (PC<sub>0-7</sub>) ← addr    If T1 = 1  
(PC) = (PC) + 2    If T1 = 0

**Example:** JT1HI: JT1 COUNT    ;JUMP TO 'COUNT' ROUTINE  
;IF T1 = 1

**JZ address    Jump If Accumulator Is Zero**

**Encoding:** 1 1 0 0    0 1 1 0    a<sub>7</sub> a<sub>6</sub> a<sub>5</sub> a<sub>4</sub>    a<sub>3</sub> a<sub>2</sub> a<sub>1</sub> a<sub>0</sub>    C6H

**Description:** This is a 2-cycle instruction. Control passes to the specified address if the accumulator contains all zeros at the time this instruction is executed.

**Operation:** (PC<sub>0-7</sub>) ← addr    If A = 0  
(PC) = (PC) + 2    If A ≠ 0

**Example:** JACCO: JZ 0A3H    ;JUMP TO LOCATION 'A3' HEX  
;IF ACC VALUE IS ZERO

**MOV A,#data    Move Immediate Data to Accumulator**

**Encoding:** 0 0 1 0    0 0 1 1    a<sub>7</sub> a<sub>6</sub> a<sub>5</sub> a<sub>4</sub>    a<sub>3</sub> a<sub>2</sub> a<sub>1</sub> a<sub>0</sub>    23H

**Description:** This is a 2-cycle instruction. The 8-bit value specified by 'data' is loaded in the accumulator.

**Operation:** (A) ← data

**Example:** MOV A,#0A3H    ;MOVE 'A3' HEX TO ACC

**MOV A,PSW    Move PSW Contents to Accumulator**

**Encoding:** 1 1 0 0    0 1 1 1    C7H

**Description:** The contents of the program status word are moved to the accumulator.

**Operation:** (A) ← (PSW)

**Example:** Jump to 'RB1SET' routine if PSW bank switch, bit 4, is set.  
BSCHK: MOV A,PSW    ;MOVE PSW CONTENTS TO ACC  
JB4 RB1SET    ;JUMP TO 'RB1SET' IF ACC BIT 4 = 1

**MOV A,R<sub>r</sub> Move Register Contents to Accumulator**

Encoding: 

1	1	1	1	1	r	r	r
---	---	---	---	---	---	---	---

 F8H-FFH

Description: 8-bits of data are removed from working register 'r' into the accumulator.

Operation:  $(A) \leftarrow (R_r)$   $r = 0-7$

Example: MAR: MOV A,R3 ;MOVE CONTENTS OF REG 3 TO ACC

**MOV A,@R<sub>i</sub> Move Data Memory Contents to Accumulator**

Encoding: 

1	1	1	1	0	0	0	i
---	---	---	---	---	---	---	---

 F0H-F1H

Description: The contents of the resident data memory location addressed by bits 0-5\*\* of register 'i' are moved to the accumulator. Register 'i' contents are unaffected.

Operation:  $(A) \leftarrow ((R_i))$   $i = 0-1$

Example: Assume R1 contains 00110110.  
MADM: MOV A,@R1 ;MOVE CONTENTS OF DATA MEM  
;LOCATION 54 TO ACC

**MOV A,T Move Timer/Counter Contents to Accumulator**

Encoding: 

0	1	0	0	0	0	1	0
---	---	---	---	---	---	---	---

 42H

Description: The contents of the timer/event-counter register are moved to the accumulator.

Operation:  $(A) \leftarrow (T)$

Example: Jump to "EXIT" routine when timer reaches '64', that is, when bit 6 set—  
assuming initialization 64,  
TIMCHK: MOV A,T ;MOVE TIMER CONTENTS TO ACC  
JB6 EXIT ;JUMP TO 'EXIT' IF ACC BIT 6 = 1

**MOV PSW,A Move Accumulator Contents to PSW**

Encoding: 

1	1	0	1	0	1	1	1
---	---	---	---	---	---	---	---

 D7H

Description: The contents of the accumulator are moved into the program status word. All condition bits and the stack pointer are affected by this move.

Operation:  $(PSW) \leftarrow (A)$

Example: Move up stack pointer by two memory locations, that is, increment the  
pointer by one.  
INCPTR: MOV A,PSW ;MOVE PSW CONTENTS TO ACC  
INC A ;INCREMENT ACC BY ONE  
MOV PSW,A ;MOVE ACC CONTENTS TO PSW

\*\* 0-5 in 8048AH/8748H  
0-6 in 8049AH/8749H  
0-7 in 8050AH



**MOV R<sub>p</sub>,A Move Accumulator Contents to Register**

Encoding: 

1	0	1	0
---	---	---	---

1	r	r	r
---	---	---	---

 A8H-AFH

Description: The contents of the accumulator are moved to register 'r'.

Operation:  $(Rr) \leftarrow (A)$  r = 0-7

Example: MRA: MOV R0,A ;MOVE CONTENTS OF ACC TO REG 0

**MOV R<sub>p</sub>,#data Move Immediate Data to Register**

Encoding: 

1	0	1	1
---	---	---	---

1	r <sub>2</sub>	r <sub>1</sub>	r <sub>0</sub>
---	----------------	----------------	----------------

d <sub>7</sub>	d <sub>6</sub>	d <sub>5</sub>	d <sub>4</sub>
----------------	----------------	----------------	----------------

d <sub>3</sub>	d <sub>2</sub>	d <sub>1</sub>	d <sub>0</sub>
----------------	----------------	----------------	----------------

 B8H-BFH

Description: This is a 2-cycle instruction. The 8-bit value specified by 'data' is moved to register 'r'.

Operation:  $(Rr) \leftarrow \text{data}$  r = 0-7

Examples: MIR4: MOV R4,#HEXTEN ;THE VALUE OF THE SYMBOL  
;HEXTEN' IS MOVED INTO REG 4  
MIR 5: MOV R5,#PI\*(R\*R) ;THE VALUE OF THE EXPRESSION  
;PI\*(R\*R)' IS MOVED INTO REG 5  
MIR 6: MOV R6, #0ADH ;'AD' HEX IS MOVED INTO REG 6

**MOV @ R<sub>i</sub>,A Move Accumulator Contents to Data Memory**

Encoding: 

1	0	1	0
---	---	---	---

0	0	0	i
---	---	---	---

 A0H-A1H

Description: The contents of the accumulator are moved to the resident data memory location whose address is specified by bits 0-5\*\* of register 'i'. Register 'i' contents are unaffected.

Operation:  $((Ri)) \leftarrow (A)$  i = 0-1

Example: Assume R0 contains 00000111.  
MDMA: MOV @R0,A ;MOVE CONTENTS OF ACC TO  
;LOCATION 7 (REG 7)

**MOV @ R<sub>i</sub>,#data Move Immediate Data to Data memory**

Encoding: 

1	0	1	1
---	---	---	---

0	0	0	i
---	---	---	---

d <sub>7</sub>	d <sub>6</sub>	d <sub>5</sub>	d <sub>4</sub>
----------------	----------------	----------------	----------------

d <sub>3</sub>	d <sub>2</sub>	d <sub>1</sub>	d <sub>0</sub>
----------------	----------------	----------------	----------------

 B0H-B1H

Description: This is a 2-cycle instruction. The 8-bit value specified by 'data' is moved to the resident data memory location addressed by register 'i', bits 0-5\*\*.

Operation:  $((Ri)) \leftarrow \text{data}$  i = 0-1

Examples: Move the hexadecimal value AC3F to locations 62-63.  
MIDM: MOV R0,#62 ;MOVE '62' DEC TO ADDR REG 0  
MOV @R0,#0ACH ;MOVE 'AC' HEX TO LOCATION 62  
INC R0 ;INCREMENT REG 0 to '63'  
MOV @R0,#3FH ;MOVE '3F' HEX TO LOCATION 63

\*\* 0-5 in 8048AH/8748H

0-6 in 8049AH/8749H

0-7 in 8050AH

**MOV T,A Move Accumulator Contents to Timer/Counter**

**Encoding:** 0 1 1 0 0 0 1 0 62H

**Description:** The contents of the accumulator are moved to the timer/event-counter register.

**Operation:** (T) ← (A)

**Example:** Initialize and start event counter.

INITEC: CLR A ;CLEAR ACC TO ZEROS  
MOV T,A ;MOVE ZEROS TO EVENT COUNTER  
START CNT ;START COUNTER

**MOVD A,Pp Move Port 4-7 Data to Accumulator**

**Encoding:** 0 0 0 0 1 1 p p 0CH-0FH

**Description:** This is a 2-cycle instruction. Data on 8243 port 'p' is moved (read) to accumulator bits 0-3. Accumulator bits 4-7 are zeroed.

**Operation:** (0-3) ← (Pp) p = 4-7  
(4-7) ← 0

**Note:** Bits 0-7 of the opcode are used to represent ports 4-7. If you are coding in binary rather than assembly language, the mapping is as follows:

Bits 1 0	Port
0 0	4
0 1	5
1 0	6
1 1	7

**Example:** INPPT5: MOVD A,P5 ;MOVE PORT 5 DATA TO ACC  
;BITS 0-3, ZERO ACC BITS 4-7

**MOVD Pp,A Move Accumulator Data to Port 4-7**

**Encoding:** 0 0 1 1 1 1 p p 3CH-3FH

**Description:** This is a 2-cycle instruction. Data in accumulator bits 0-3 is moved (written) to 8243 port 'p'. Accumulator bits 4-7 are unaffected. (See NOTE above regarding port mapping.)

**Operation:** (Pp) ← (A<sub>0-3</sub>) P = 4-7

**Example:** Move data in accumulator to ports 4 and 5.

OUTP45: MOVD P4,A ;MOVE ACC BITS 0-3 TO PORT 4  
SWAP A ;EXCHANGE ACC BITS 0-3 and 4-7  
MOVD P5,A ;MOVE ACC BITS 0-3 TO PORT 5

## MOVP A,@A Move Current Page Data to Accumulator

Encoding: 1 0 1 0 0 0 1 1 A3H

**Description:** The contents of the program memory location addressed by the accumulator are moved to the accumulator. Only bits 0-7 of the program counter are affected, limiting the program memory reference to the current page. The program counter is restored *following* this operation.

**Operation:**  $(PC_{0-7}) \leftarrow (A)$   
 $(A) \leftarrow ((PC))$

Note: This is a 1-byte, 2-cycle instruction. If it appears in location 255 of a program memory page, @A addresses a location in the *following* page.

**Example:** MOV128: MOV A,#128 ;MOVE '128' DEC TO ACC  
MOVP A,@A ;CONTENTS OF 129th LOCATION IN  
;CURRENT PAGE ARE MOVED TO ACC

## MOVP3 A,@A Move Page 3 Data to Accumulator

Encoding: 1 1 1 0 0 0 1 1 E3H

**Description:** This is a 2-cycle instruction. The contents of the program memory location (within page 3) addressed by the accumulator are moved to the accumulator. The program counter is restored following this operation.

**Operation:**  $(PC_{0-7}) \leftarrow (A)$   
 $(PC_{8-11}) \leftarrow 0011$   
 $(A) \leftarrow ((PC))$

**Example:** Look up ASCII equivalent of hexadecimal code in table contained at the beginning of page 3. Note that ASCII characters are designated by a 7-bit code; the eighth bit is always reset.

TABSCH: MOV A,#0B8H ;MOVE 'B8' HEX TO ACC (10111000)  
ANL A,#7FHT ;LOGICAL AND ACC TO MASK BIT  
;7 (00111000)  
MOVP3 A,@A ;MOVE CONTENTS OF LOCATION '38'  
;HEX IN PAGE 3 TO ACC (ASCII '8')

Access contents of location in page 3 labelled TAB1.

Assume current program location is not in page 3.

TABSCH: MOV A,#LOW TAB 1 ;ISOLATE BITS 0-7 OF LABEL  
;ADDRESS VALUE  
MOVP3 A,@A ;MOVE CONTENTS OF PAGE 3  
;LOCATION LABELED 'TAB1' TO ACC

## MCS®-48 INSTRUCTION SET

### MOVX A,@R<sub>i</sub> Move External-Data-Memory Contents to Accumulator

**Encoding:**

1	0	0	0	0	0	0	i
---	---	---	---	---	---	---	---

 80H-81H

**Description:** This is a 2-cycle instruction. The contents of the external data memory location addressed by register 'i' are moved to the accumulator. Register 'i' contents are unaffected. A read pulse is generated.

**Operation:** (A) ← ((R<sub>i</sub>)) i = 0-1

**Example:** Assume R1 contains 01110110.

MAXDM: MOVX A,@R1 ;MOVE CONTENTS OF LOCATION  
;118 TO ACC

### MOVX @R<sub>i</sub>,A Move Accumulator Contents to External Data Memory

**Encoding:**

1	0	0	1	0	0	0	i
---	---	---	---	---	---	---	---

 90H-91H

**Description:** This is a 2-cycle instruction. The contents of the accumulator are moved to the external data memory location addressed by register 'i'. Register 'i' contents are unaffected. A write pulse is generated.

**Operation:** ((R<sub>i</sub>)) ← A i = 0-1

**Example:** Assume R0 contains 11000111.

MXDMA: MOVX @R0,A ;MOVE CONTENTS OF ACC TO  
;LOCATION 199 IN EXPANDED  
;DATA MEMORY

### NOP The NOP Instruction

**Encoding:**

0	0	0	0	0	0	0	0
---	---	---	---	---	---	---	---

 00H

**Description:** No operation is performed. Execution continues with the following instruction.

### ORL A,R<sub>r</sub> Logical OR Accumulator With Register Mask

**Encoding:**

0	1	0	0	1	r	r	r
---	---	---	---	---	---	---	---

 48H-4FH

**Description:** Data in the accumulator is logically ORed with the mask contained in working register 'r'.

**Operation:** (A) ← (A) OR (R<sub>r</sub>) r = 0-7

**Example:** ORREG: ORL A,R4

;OR' ACC CONTENTS WITH  
;MASK IN REG 4



**ORL A,@R<sub>i</sub> Logical OR Accumulator With Memory Mask****Encoding:**

0	1	0	0	0	0	0	i
---	---	---	---	---	---	---	---

 40H-41H**Description:** Data in the accumulator is logically ORed with the mask contained in the resident data memory location referenced by register "i", bits 0-5\*\*.**Operation:** (A) ← (A) OR ((R<sub>i</sub>)) i = 0-1**Example:** ORDM: MOV R0,#3FH ;MOVE '3F' HEX TO REG 0  
ORL A,@R0 ;'OR' AC CONTENTS WITH MASK  
;IN LOCATION 63**ORL A,#data Logical OR Accumulator With Immediate Mask****Encoding:**

0	1	0	0	0	0	1	1
---	---	---	---	---	---	---	---

d <sub>7</sub>	d <sub>6</sub>	d <sub>5</sub>	d <sub>4</sub>	d <sub>3</sub>	d <sub>2</sub>	d <sub>1</sub>	d <sub>0</sub>
----------------	----------------	----------------	----------------	----------------	----------------	----------------	----------------

 43H**Description:** This is a 2-cycle instruction. Data in the accumulator is logically ORed with an immediately-specified mask.**Operation:** (A) ← (A) OR data**Example:** ORID: ORL A,#'X' ;'OR' ACC CONTENTS WITH MASK  
;01011000 (ASCII VALUE OF 'X')**ORL BUS,#data\* Logical OR BUS With Immediate Mask****Encoding:**

1	0	0	0	1	0	0	0
---	---	---	---	---	---	---	---

d <sub>7</sub>	d <sub>6</sub>	d <sub>5</sub>	d <sub>4</sub>	d <sub>3</sub>	d <sub>2</sub>	d <sub>1</sub>	d <sub>0</sub>
----------------	----------------	----------------	----------------	----------------	----------------	----------------	----------------

 88H**Description:** This is a 2-cycle instruction. Data on the BUS port is logically ORed with an immediately-specified mask. This instruction assumes prior specification on an 'OUTL BUS,A' instruction.**Operation:** (BUS) ← (BUS) OR data**Example:** ORBUS: ORL BUS,#HEXMSK ;'OR' BUS CONTENTS WITH MASK  
;EQUAL VALUE OF SYMBOL 'HEXMSK'**ORL Pp, #data Logical OR Port 1 or 2 With Immediate Mask****Encoding:**

1	0	0	0	1	0	p	p
---	---	---	---	---	---	---	---

d <sub>7</sub>	d <sub>6</sub>	d <sub>5</sub>	d <sub>4</sub>	d <sub>3</sub>	d <sub>2</sub>	d <sub>1</sub>	d <sub>0</sub>
----------------	----------------	----------------	----------------	----------------	----------------	----------------	----------------

 89H-8AH**Description:** This is a 2-cycle instruction. Data on port 'p' is logically ORed with an immediately-specified mask.**Operation:** (Pp) ← (Pp) OR data p = 1-2**Example:** ORP1: ORL P1, #0FFH ;'OR' PORT 1 CONTENTS WITH MASK  
;'FF' HEX (SET PORT 1 TO ALL ONES)

\* For use with internal program memory ONLY.

\*\* 0-5 in 8048AH/8748H

0-6 in 8049AH/8749H

0-7 in 8050AH

**ORLD Pp,A Logical OR Port 4-7 With Accumulator Mask****Encoding:** 1 0 0 0 1 1 p p 8CH-8FH**Description:** This is a 2-cycle instruction. Data on port 'p' is logically ORed with the digit mask contained in accumulator bits 0-3.**Operation:** (Pp) ← (Pp) OR (A<sub>0-3</sub>) p = 4-7**Example:** ORP7: ORLD P7,A ;'OR' PORT 7 CONTENTS WITH ACC  
;BITS 0-3**OUTL BUS,A\* Output Accumulator Data to BUS****Encoding:** 0 0 0 0 0 0 1 0 02H**Description:** This is a 2-cycle instruction. Data residing in the accumulator is transferred (written) to the BUS port and latched. The latched data remains valid until altered by another OUTL instruction. Any other instruction requiring use of the BUS port (except INS) destroys the contents of the BUS latch. This includes expanded memory operations (such as the MOVX instruction). Logical operations on BUS data (AND, OR) assume the OUTL BUS,A instruction has been issued previously.**Operation:** (BUS) ← (A)**Example:** OUTLBP: OUTL BUS, A ;OUTPUT ACC CONTENTS TO BUS**OUTL Pp,A Output Accumulator Data to Port 1 or 2****Encoding:** 0 0 1 1 1 0 p p 39H-3AH**Description:** This is a 2-cycle instruction. Data residing in the accumulator is transferred (written) to port 'p' and latched.**Operation:** (Pp) ← (A) p = 1-2**Example:** OUTLP: MOV A,R7 ;MOVE REG 7 CONTENTS TO ACC  
OUTL P2,A ;OUTPUT ACC CONTENTS TO PORT 2  
MOV A, R6 ;MOV REG 6 CONTENTS TO ACC  
OUTL P1,A ;OUTPUT ACC CONTENTS TO PORT 1

\* For use with internal program memory ONLY.

**RET Return Without PSW Restore**

**Encoding:**

1	0	0	0	0	0	1	1
---	---	---	---	---	---	---	---

 83H

**Description:** This is a 2-cycle instruction. The stack pointer (PSW bits 0-2) is decremented. The program counter is then restored from the stack. PSW bits 4-7 are not restored.

**Operation:** (SP)  $\leftarrow$  (SP)-1  
(PC)  $\leftarrow$  ((SP))

**RETR Return with PSW Restore**

**Encoding:**

1	0	0	1	0	0	1	1
---	---	---	---	---	---	---	---

 93H

**Description:** This is a 2-cycle instruction. The stack pointer is decremented. The program counter and bits 4-7 of the PSW are then restored from the stack. Note that RETR should be used to return from an interrupt, but should not be used within the interrupt service routine as it signals the end of an interrupt routine by resetting the Interrupt in Progress flip-flop.

**Operation:** (SP)  $\leftarrow$  (SP)-1  
(PC)  $\leftarrow$  ((SP))  
(PSW 4-7)  $\leftarrow$  ((SP))

**RL A Rotate Left without Carry****Encoding:** 1 1 1 0 0 1 1 1 E7H**Description:** The contents of the accumulator are rotated left one bit. Bit 7 is rotated into the bit 0 position.**Operation:**  $(A_{n+1}) \leftarrow (A_n)$   
 $(A_0) \leftarrow (A_7)$   $n = 0-6$ **Example:** Assume accumulator contains 10110001.

RLNC: RL A ;NEW ACC CONTENTS ARE 01100011

**RLC A Rotate Left through Carry****Encoding:** 1 1 1 1 0 1 1 1 F7H**Description:** The contents of the accumulator are rotated left one bit. Bit 7 replaces the carry bit; the carry bit is rotated into the bit 0 position.**Operation:**  $(A_{n+1}) \leftarrow (A_n)$   
 $n = 0-6$   
 $(A_0) \leftarrow (C)$   
 $(C) \leftarrow (A_7)$ **Example:** Assume accumulator contains a 'signed' number; isolate sign without changing value.

RLTC: CLR C

RLC A

RR A

;CLEAR CARRY TO ZERO

;ROTATE ACC LEFT, SIGN

;BIT (7) IS PLACED IN CARRY

;ROTATE ACC RIGHT — VALUE

;(BITS 0-6) IS RESTORED,

;CARRY UNCHANGED, BIT 7

;IS ZERO

**RR A Rotate Right without Carry****Encoding:** 0 1 1 1 0 1 1 1 77H**Description:** The contents of the accumulator are rotated right one bit. Bit 0 is rotated into the bit 7 position.**Operation:**  $(A_n) \leftarrow (A_{n+1})$   
 $(A_7) \leftarrow (A_0)$   $n = 0-6$ **Example:** Assume accumulator contains 10110001.

RRNC: RR A

;NEW ACC CONTENTS ARE 11011000



**RRC A Rotate Right through Carry****Encoding:** 0 1 1 0 0 1 1 1 67H**Description:** The contents of the accumulator are rotated right one bit. Bit 0 replaces the carry bit; the carry bit is rotated into the bit 7 position.**Operation:**  $(A_n) \leftarrow (A_n + 1)$   $n = 0-6$   
 $(A_7) \leftarrow (C)$   
 $(C) \leftarrow (A_0)$ **Example:** Assume carry is not set and accumulator contains 10110001.  
RRTC: RRC A ;CARRY IS SET AND ACC  
;CONTAINS 01011000**SEL MB0 Select Memory Bank 0****Encoding:** 1 1 1 0 0 1 0 1 E5H**Description:** PC bit 11 is set to zero on next JMP or CALL instruction. All references to program memory addresses fall within the range 0-2047.**Operation:**  $(DBF) \leftarrow 0$ **Example:** Assume program counter contains 834 Hex.SEL MB0 ;SELECT MEMORY BANK 0  
JMP \$+20 ;JUMP TO LOCATION 58 HEX**SEL MB1 Select Memory Bank 1****Encoding:** 1 1 1 1 0 1 0 1 F5H**Description:** PC bit 11 is set to one on next JMP or CALL instruction. All references to program memory addresses fall within the range 2048-4095.**Operation:**  $(DBF) \leftarrow 1$

## MCS®-48 INSTRUCTION SET

### SEL RB0 Select Register Bank 0

**Encoding:**

1	1	0	0
0	1	0	1

 C5H

**Description:** PSW bit 4 is set to zero. References to working registers 0-7 address data memory locations 0-7. This is the recommended setting for normal program execution.

**Operation:** (BS) ← 0

### SEL RB1 Select Register Bank 1

**Encoding:**

1	1	0	1
0	1	0	1

 D5H

**Description:** PSW bit 4 is set to one. References to working registers 0-7 address data memory locations 24-31. This is the recommended setting for interrupt service routines, since locations 0-7 are left intact. The setting of PSW bit 4 in effect at the time of an interrupt is restored by the RETR instruction when the interrupt service routine is completed.

**Operation:** (BS) ← 1

**Example:** Assume an external interrupt has occurred, control has passed to program memory location 3, and PSW bit 4 was zero before the interrupt.

<b>Operation:</b> LOC3: JN1 INIT  INIT: MOV R7,A SEL RB1 MOV R7,#0FAH	;JUMP TO ROUTINE 'INIT' IF ;INTERRUPT INPUT IS ZERO ;MOVE ACC CONTENTS TO ;LOCATION 7 ;SELECT REG BANK 1 ;MOVE 'FA' HEX TO LOCATION 31
---	---

SEL RB0 MOV A,R7 RETR	;SELECT REG BANK 0 ;RESTORE ACC FROM LOCATION 7 ;RETURN — RESTORE PC AND PSW
-----------------------------	--

### STOP TCNT Stop Timer/Event-Counter

**Encoding:**

0	1	1	0
0	1	0	1

 65H

**Description:** This instruction is used to stop both time accumulation and event counting.

## MCS®-48 INSTRUCTION SET

**Example:** Disable interrupt, but jump to interrupt routine after eight overflows and stop timer. Count overflows in register 7.

START: DIS TCNTI	;DISABLE TIMER INTERRUPT
CLR A	;CLEAR ACC TO ZEROS
MOV T,A	;MOVE ZEROS TO TIMER
MOV R7,A	;MOVE ZEROS TO REG 7
STRT T	;START TIMER
MAIN: JTF COUNT	;JUMP TO ROUTINE 'COUNT'
	;IF TF = 1 AND CLEAR TIMER FLAG
JMP MAIN	;CLOSE LOOP
COUNT: INC R7	;INCREMENT REG 7
MOV A,R7	;MOVE REG 7 CONTENTS TO ACC
JB3 INT	;JUMP TO ROUTINE 'INT' IF ACC
	;BIT 3 IS SET (REG 7 = 8)
JMP MAIN	;OTHERWISE RETURN TO ROUTINE
	MAIN

INT: STOP TCNT	;STOP TIMER
JMP 7H	;JUMP TO LOCATION 7 (TIMER)
	INTERRUPT ROUTINE

### STRT CNT Start Event Counter

**Encoding:** 0 1 0 0 0 1 0 1 45H

**Description:** The test 1 (T1) pin is enabled as the event-counter input and the counter is started. The event-counter register is incremented with each high-to-low transition on the T1 pin.

**Example:** Initialize and start event counter. Assume overflow is desired with first T1 input.

STARTC: EN TCNTI	;ENABLE COUNTER INTERRUPT
MOV A,#0FFH	;MOVE 'FF'HEX (ONES) TO ACC
MOV T,A	;MOVES ONES TO COUNTER
STRT CNT	;ENABLE T1 AS COUNTER
	;INPUT AND START

**STRT T Start Timer**

Encoding: 

0	1	0	1
---	---	---	---

0	1	0	1
---	---	---	---

 55H

**Description:** Timer accumulation is initiated in the timer register. The register is incremented every 32 instruction cycles. The prescaler which counts the 32 cycles is cleared but the timer register is not.

**Example:** Initialize and start timer.

```

STARTT: CLR A           ;CLEAR ACC TO ZEROS
        MOV T,A         ;MOVE ZEROS TO TIMER
        EN TCNTI        ;ENABLE TIMER INTERRUPT
        STRT T          ;START TIMER

```

**SWAP A Swap Nibbles within Accumulator**

Encoding: 

0	1	0	0
---	---	---	---

0	1	1	1
---	---	---	---

 47H

**Description:** Bits 0-3 of the accumulator are swapped with bits 4-7 of the accumulator.

**Operation:**  $(A_{4-7}) \rightleftharpoons (A_{0-3})$

**Example:** Pack bits 0-3 of locations 50-51 into location 50.

```

PCKDIG: MOV R0, #50      ;MOVE '50' DEC TO REG 0
        MOV R1, #51      ;MOVE '51' DEC TO REG 1
        XCHD A,@R0       ;EXCHANGE BITS 0-3 OF ACC
                           ;AND LOCATION 50
        SWAP A           ;SWAP BITS 0-3 AND 4-7 OF ACC
        XCHD A,@R1       ;EXCHANGE BITS 0-3 OF ACC AND
                           ;LOCATION 51
        MOV @R0,A        ;MOVE CONTENTS OF ACC TO
                           ;LOCATION 50

```

**XCH A,R<sub>r</sub> Exchange Accumulator-Register Contents**

Encoding: 

0	0	1	0
---	---	---	---

1	r	r	r
---	---	---	---

 28H-2FH

**Description:** The contents of the accumulator and the contents of working register 'r' are exchanged.

**Operation:**  $(A) \rightleftharpoons (R_r)$   $r = 0-7$

**Example:** Move PSW contents to Reg 7 without losing accumulator contents.

```

XCHAR7: XCH A,R7        ;EXCHANGE CONTENTS OF REG 7
                           ;AND ACC
        MOV A, PSW      ;MOVE PSW CONTENTS TO ACC
        XCH A,R7        ;EXCHANGE CONTENTS OF REG 7
                           ;AND ACC AGAIN

```



**XCH A,@R<sub>i</sub> Exchange Accumulator and Data Memory Contents****Encoding:** 0 0 1 0 | 0 0 0 i 20H-21H**Description:** The contents of the accumulator and the contents of the resident data memory location addressed by bits 0-5\*\* of register 'i' are exchanged. Register 'i' contents are unaffected.**Operation:** (A)  $\leftrightarrow$  ((R<sub>i</sub>)) i = 0-1**Example:** Decrement contents of location 52.

```

DEC52: MOV R0,#52      ;MOVE '52' DEC TO ADDRESS REG 0
        XCH A,@R0      ;EXCHANGE CONTENTS OF ACC
                        ;AND LOCATION 52
        DEC A           ;DECREMENT ACC CONTENTS
        XCH A,@R0      ;EXCHANGE CONTENTS OF ACC
                        ;AND LOCATION 52 AGAIN

```

**XCHD A,@R<sub>i</sub> Exchange Accumulator and Data Memory 4-Bit Data****Encoding:** 0 0 1 1 | 0 0 0 i 30H-31H**Description:** This instruction exchanges bits 0-3 of the accumulator with bits 0-3 of the data memory location addressed by bits 0-5\*\* of register 'i'. Bits 4-7 of the accumulator, bits 4-7 of the data memory location, and the contents of register 'i' are unaffected.**Operation:** (A<sub>0-3</sub>)  $\leftrightarrow$  ((R<sub>i</sub>0-3)) i = 0-1**Example:** Assume program counter contents have been stacked in locations 22-23.

```

XCHNIB: MOV R0,#23      ;MOVE '23' DEC TO REG 0
        CLR A           ;CLEAR ACC TO ZEROS
        XCHD A,@R0      ;EXCHANGE BITS 0-3 OF ACC AND
                        ;LOCATION 23 (BTS 8-11 OF PC ARE
                        ;ZEROED, ADDRESS REFERS
                        ;TO PAGE 0)

```

**XRL A,R<sub>r</sub> Logical XOR Accumulator With Register Mask****Encoding:** 1 1 0 1 | 1 r r r D8H-DFH**Description:** Data in the accumulator is EXCLUSIVE ORed with the mask contained in working register 'r'.**Operation:** (A)  $\leftarrow$  (A) XOR (R<sub>r</sub>) r = 0-7**Example:** XORREG: XRL A,R5 ;'XOR' ACC CONTENTS WITH  
;MASK IN REG 5

\*\* 0-5 in 8048AH/8748H  
 0-6 in 8049AH/8749H  
 0-7 in 8050AH

**XRL A,@R<sub>i</sub> Logical XOR Accumulator With Memory Mask**

**Encoding:**

1	1	0	1	0	0	0	i
---	---	---	---	---	---	---	---

 D0H-D1H

**Description:** Data in the accumulator is EXCLUSIVE ORed with the mask contained in the data memory location addressed by register 'i', bits 0-5.\*\*

**Operation:**  $(A) \leftarrow (A) \text{ XOR } ((R_i))$  i = 0-1

**Example:** XORDM: MOV R1,#20H ;MOVE '20' HEX TO REG 1  
XRL A,@R1 ;'XOR' ACC CONTENTS WITH MASK  
;IN LOCATION 32

**XRL A,#data Logical XOR Accumulator With Immediate Mask**

**Encoding:**

1	1	0	1	0	0	1	1
---	---	---	---	---	---	---	---

d <sub>7</sub>	d <sub>6</sub>	d <sub>5</sub>	d <sub>4</sub>	d <sub>3</sub>	d <sub>2</sub>	d <sub>1</sub>	d <sub>0</sub>
----------------	----------------	----------------	----------------	----------------	----------------	----------------	----------------

 D3H

**Description:** This is a 2-cycle instruction. Data in the accumulator is EXCLUSIVE ORed with an immediately-specified mask.

**Operation:**  $(A) \leftarrow (A) \text{ XOR } \text{data}$

**Example:** XORID: XOR A,#HEXTEN ;XOR CONTENTS OF ACC WITH MASK  
;EQUAL VALUE OF SYMBOL 'HEXTEN'

\*\* 0-5 in 8048AH/8748H  
0-6 in 8049AH/8749H  
0-7 in 8050AH



---



15

MCS® 48 Data Sheets

# 8243 MCS®-48 INPUT/OUTPUT EXPANDER

■ 0° C to 70° C Operation

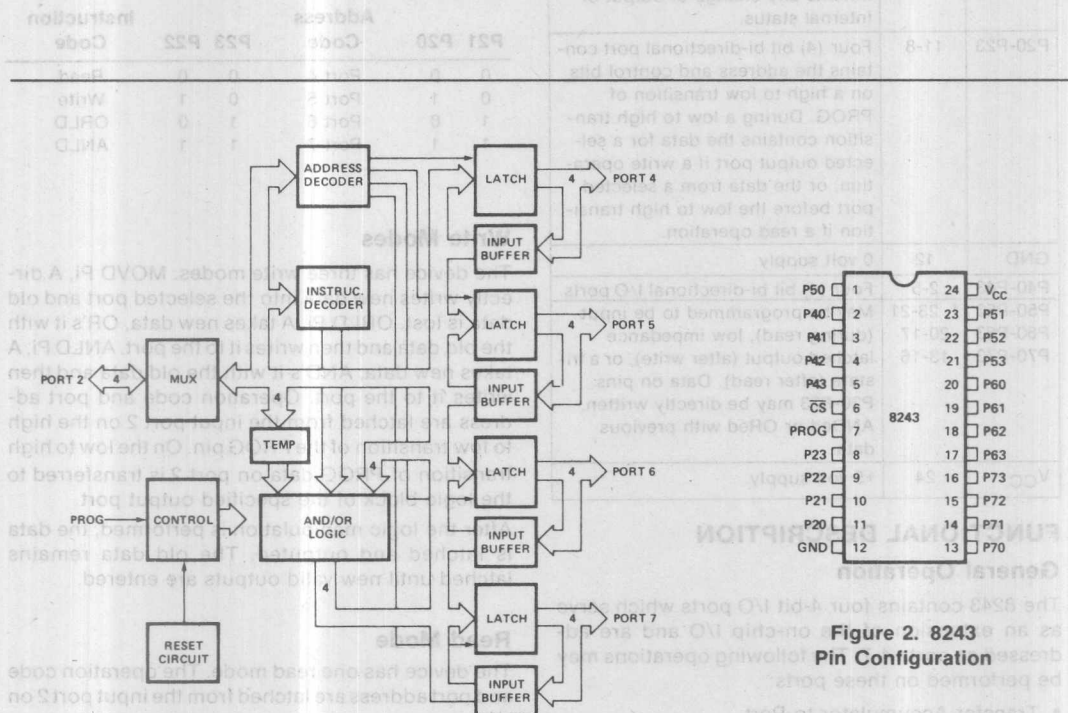


Figure 1. 8243  
Block Diagram

Figure 2. 8243  
Pin Configuration

Table 1. Pin Description

Symbol	Pin No.	Function
PROG	7	Clock Input. A high to low transition on PROG signifies that address and control are available on P20-P23, and a low to high transition signifies that data is available on P20-P23.
CS	6	Chip Select Input. A high on CS inhibits any change of output or internal status.
P20-P23	11-8	Four (4) bit bi-directional port contains the address and control bits on a high to low transition of PROG. During a low to high transition contains the data for a selected output port if a write operation, or the data from a selected port before the low to high transition if a read operation.
GND	12	0 volt supply.
P40-P43	2-5	Four (4) bit bi-directional I/O ports.
P50-P53	1, 23-21	May be programmed to be input (during read), low impedance latched output (after write), or a tri-state (after read). Data on pins P20-P23 may be directly written, ANDed or ORed with previous data.
P60-P63	20-17	
P70-P73	13-16	
VCC	24	+5 volt supply.

## FUNCTIONAL DESCRIPTION

### General Operation

The 8243 contains four 4-bit I/O ports which serve as an extension of the on-chip I/O and are addressed as ports 4-7. The following operations may be performed on these ports:

- Transfer Accumulator to Port.
- Transfer Port to Accumulator.
- AND Accumulator to Port.
- OR Accumulator to Port.

All communication between the 8048 and the 8243 occurs over Port 2 (P20-P23) with timing provided by an output pulse on the PROG pin of the processor. Each transfer consists of two 4-bit nibbles:

The first containing the "op code" and port address and the second containing the actual 4-bits of data. A high to low transition of the PROG line indicates that address is present while a low to high transition indicates the presence of data. Additional 8243's may be added to the 4-bit bus and chip selected using additional output lines from the 8048/8748/8035.

### Power On Initialization

Initial application of power to the device forces input/output ports 4, 5, 6, and 7 to the tri-state and port 2 to the input mode. The PROG pin may be either high or low when power is applied. The first high to low transition of PROG causes device to exit power on mode. The power on sequence is initiated if VCC drops below 1V.

Address			Instruction		
P21	P20	Code	P23	P22	Code
0	0	Port 4	0	0	Read
0	1	Port 5	0	1	Write
1	0	Port 6	1	0	ORLD
1	1	Port 7	1	1	ANLD

### Write Modes

The device has three write modes. MOVD Pi, A directly writes new data into the selected port and old data is lost. ORLD Pi, A takes new data, OR's it with the old data and then writes it to the port. ANLD Pi, A takes new data, AND's it with the old data and then writes it to the port. Operation code and port address are latched from the input port 2 on the high to low transition of the PROG pin. On the low to high transition of PROG data on port 2 is transferred to the logic block of the specified output port.

After the logic manipulation is performed, the data is latched and outputted. The old data remains latched until new valid outputs are entered.

### Read Mode

The device has one read mode. The operation code and port address are latched from the input port 2 on the high to low transition of the PROG pin. As soon as the read operation and port address are decoded, the appropriate outputs are tri-stated, and the input buffers switched on. The read operation is terminated by a low to high transition of the PROG pin. The port (4, 5, 6 or 7) that was selected is switched to the tri-stated mode while port 2 is returned to the input mode.

Normally, a port will be in an output (write mode) or input (read mode). If modes are changed during operation, the first read following a write should be ignored; all following reads are valid. This is to allow the external driver on the port to settle after the first read instruction removes the low impedance drive from the 8243 output. A read of any port will leave that port in a high impedance state.

**ABSOLUTE MAXIMUM RATINGS\***

Ambient Temperature Under Bias ..... 0°C to 70°C  
 Storage Temperature ..... -65°C to +150°C  
 Voltage on Any Pin  
   With Respect to Ground ..... -0.5 V to +7V  
 Power Dissipation ..... 1 Watt

*\*NOTICE: Stresses above those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.*

**D.C. CHARACTERISTICS** ( $T_A = 0^\circ\text{C}$  to  $70^\circ\text{C}$ ,  $V_{CC} = 5V \pm 10\%$ )

Symbol	Parameter	Min	Typ	Max	Units	Test Conditions
VIL	Input Low Voltage	-0.5		0.8	V	
VIH	Input High Voltage	2.0		$V_{CC}+0.5$	V	
VOL1	Output Low Voltage Ports 4-7			0.45	V	$I_{OL} = 4.5\text{ mA}^*$
VOL2	Output Low Voltage Port 7			1	V	$I_{OL} = 20\text{ mA}$
VOH1	Output High Voltage Ports 4-7	2.4			V	$I_{OH} = 240\mu\text{A}$
IIL1	Input Leakage Ports 4-7	-10		20	$\mu\text{A}$	$V_{in} = V_{CC}$ to OV
IIL2	Input Leakage Port 2, CS, PROG	-10		10	$\mu\text{A}$	$V_{in} = V_{CC}$ to OV
VOL3	Output Low Voltage Port 2			0.45	V	$I_{OL} = 0.6\text{ mA}$
ICC	$V_{CC}$ Supply Current		10	20	mA	Note 1
VOH2	Output Voltage Port 2	2.4				$I_{OH} = 100\mu\text{A}$
IOL	Sum of all $I_{OL}$ from 16 Outputs			72	mA	4.5 mA Each Pin

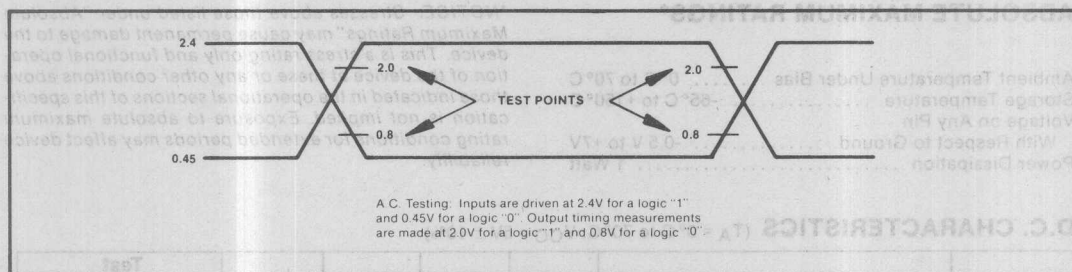
\*See following graph for additional sink current capability

**A.C. CHARACTERISTICS** ( $T_A = 0^\circ\text{C}$  to  $70^\circ\text{C}$ ,  $V_{CC} = 5V \pm 10\%$ )

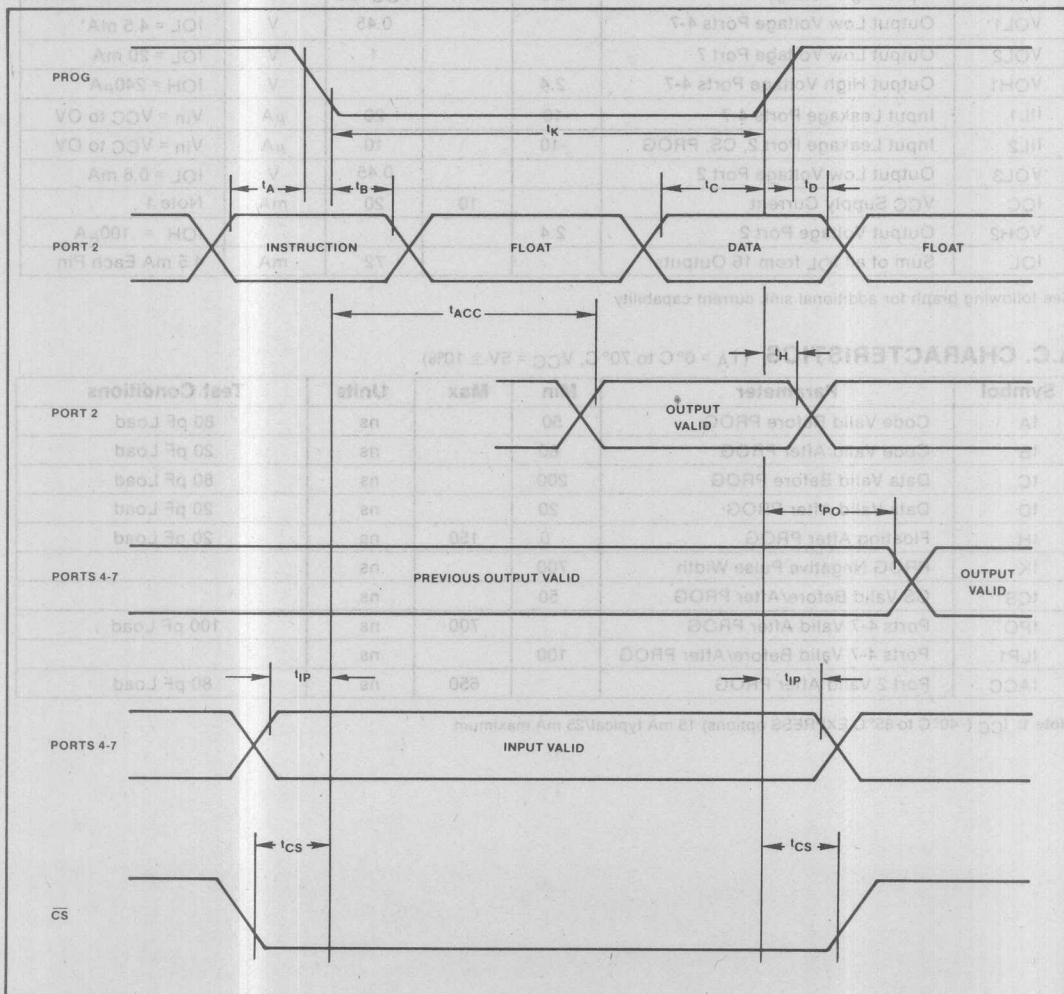
Symbol	Parameter	Min	Max	Units	Test Conditions
tA	Code Valid Before PROG	50		ns	80 pF Load
tB	Code Valid After PROG	60		ns	20 pF Load
tC	Data Valid Before PROG	200		ns	80 pF Load
tD	Data Valid After PROG	20		ns	20 pF Load
tH	Floating After PROG	0	150	ns	20 pF Load
tK	PROG Negative Pulse Width	700		ns	
tCS	CS Valid Before/After PROG	50		ns	
tPO	Ports 4-7 Valid After PROG		700	ns	100 pF Load
tLP1	Ports 4-7 Valid Before/After PROG	100		ns	
tACC	Port 2 Valid After PROG		650	ns	80 pF Load

**Note 1:**  $I_{CC}$  ( $-40^\circ\text{C}$  to  $85^\circ\text{C}$  EXPRESS options) 15 mA typical/25 mA maximum.





# WAVEFORMS



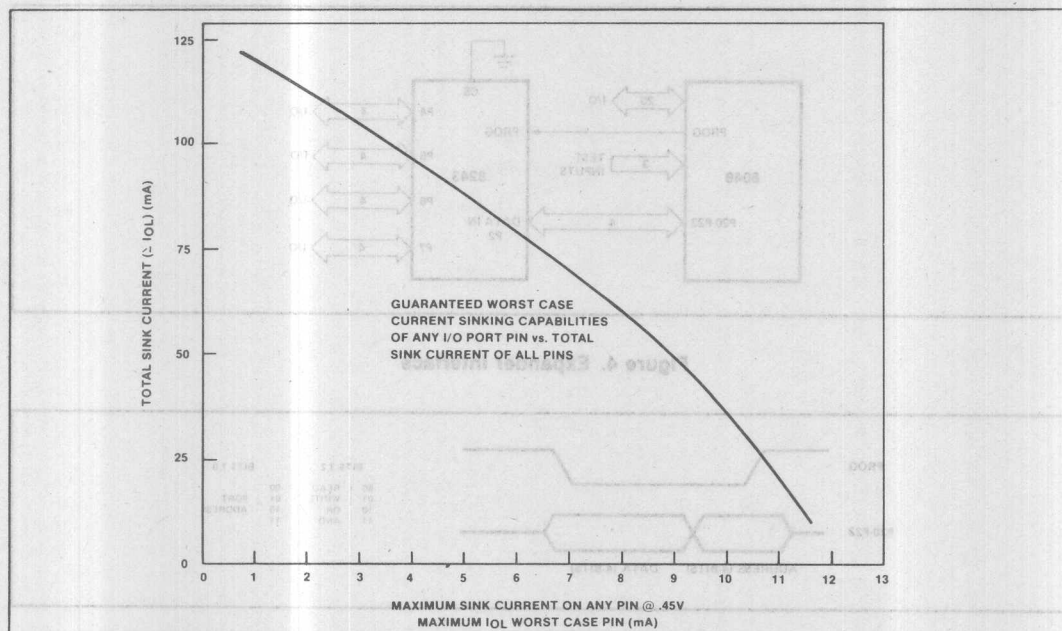


Figure 3

## Sink Capability

The 8243 can sink 5 mA @ .45V on each of its 16 I/O lines simultaneously. If, however, all lines are not sinking simultaneously or all lines are not fully loaded, the drive capability of any individual line increases as is shown by the accompanying curve.

For example, if only 5 of the 16 lines are to sink current at one time, the curve shows that each of those 5 lines is capable of sinking 9 mA @ .45V (if any lines are to sink 9 mA the total IOL must not exceed 45 mA or five 9 mA loads).

Example: How many pins can drive 5 TTL loads (1.6 mA) assuming remaining pins are unloaded?

$$IOL = 5 \times 1.6 \text{ mA} = 8 \text{ mA}$$

$$\epsilon IOL = 60 \text{ mA from curve}$$

$$\# \text{ pins} = 60 \text{ mA} \div 8 \text{ mA/pin} = 7.5 = 7$$

In this case, 7 lines can sink 8 mA for a total of 56 mA. This leaves 4 mA sink current capability which can be divided in any way among the remaining 8 I/O lines of the 8243.

Example: This example shows how the use of the 20 mA sink capability of Port 7 affects the sinking capability of the other I/O lines.

An 8243 will drive the following loads simultaneously.

2 loads—20 mA @ 1V (port 7 only)

8 loads—4 mA @ .45V

6 loads—3.2 mA @ .45V

Is this within the specified limits?

$\epsilon IOL = (2 \times 20) + (8 \times 4) + (6 \times 3.2) = 91.2 \text{ mA}$ .  
From the curve: for IOL = 4 mA,  $\epsilon IOL \approx 93 \text{ mA}$ .  
since  $91.2 \text{ mA} < 93 \text{ mA}$  the loads are within specified limits.

Although the 20 mA @ 1V loads are used in calculating  $\epsilon IOL$ , it is the largest current required @ .45V which determines the maximum allowable  $\epsilon IOL$ .

**NOTE:** A10 to 50K  $\Omega$  pullup resistor to +5V should be added to 8243 outputs when driving to 5V CMOS directly.

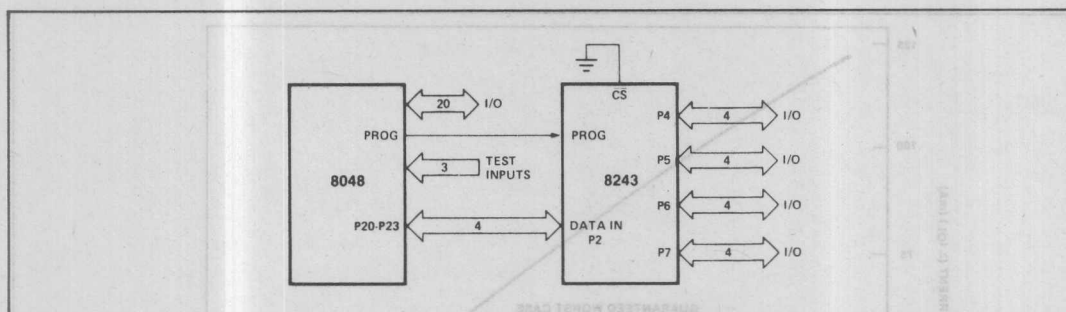


Figure 4. Expander Interface

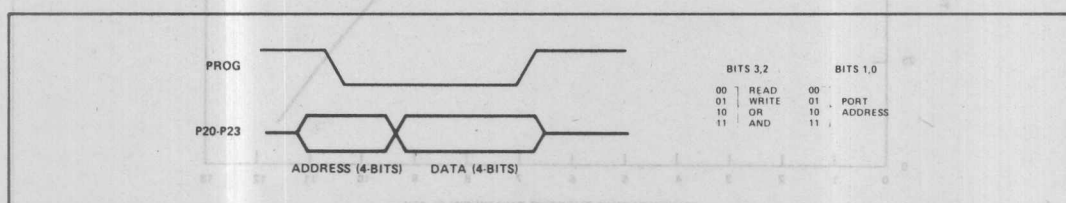


Figure 5. Output Expander Timing

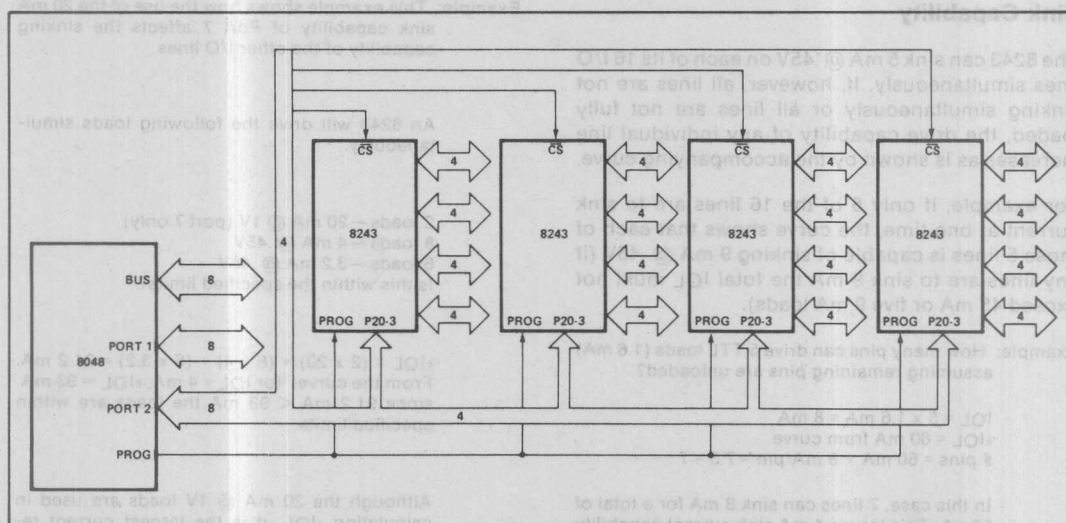


Figure 6. Using Multiple 8243's

# 8048AH/8035AHL/8049AH 8039AHL/8050AH/8040AHL HMOS SINGLE-COMPONENT 8-BIT MICROCOMPUTER

- High Performance HMOS II
- Interval Timer/Event Counter
- Two Single Level Interrupts
- Single 5-Volt Supply
- Over 96 Instructions; 90% Single Byte

- Reduced Power Consumption
- Compatible with 8080/8085 Peripherals
- Easily Expandable Memory and I/O
- Up to 1.36  $\mu$ Sec Instruction Cycle
- All Instructions 1 or 2 cycles

The Intel MCS®-48 family are totally self-sufficient, 8-bit parallel computers fabricated on single silicon chips using Intel's advanced N-channel silicon gate HMOS process.

The family contains 27 I/O lines, an 8-bit timer/counter, and on-board oscillator/clock circuits. For systems that require extra capability, the family can be expanded using MCS®-80/MCS®-85 peripherals.

To minimize development problems and provide maximum flexibility, a logically and functionally pin-compatible version of the ROM devices with UV-erasable user-programmable EPROM program memory is available with minor differences.

These microcomputers are designed to be efficient controllers as well as arithmetic processors. They have extensive bit handling capability as well as facilities for both binary and BCD arithmetic. Efficient use of program memory results from an instruction set consisting mostly of single byte instructions and no instructions over 2 bytes in length.

Device	Internal Memory		RAM Standby
8050AH	4K $\times$ 8 ROM	256 $\times$ 8 RAM	yes
8049AH	2K $\times$ 8 ROM	128 $\times$ 8 RAM	yes
8048AH	1K $\times$ 8 ROM	64 $\times$ 8 RAM	yes
8040AHL	none	256 $\times$ 8 RAM	yes
8039AHL	none	128 $\times$ 8 RAM	yes
8035AHL	none	64 $\times$ 8 RAM	yes

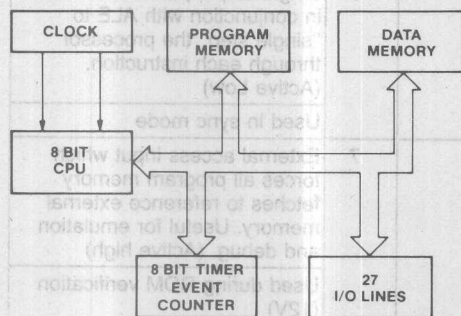


Figure 1.  
Block Diagram

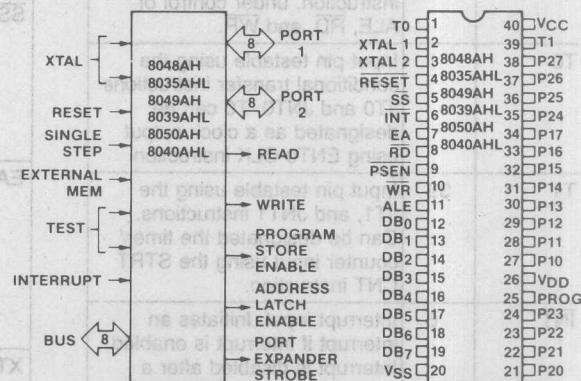


Figure 2.  
Logic Symbol

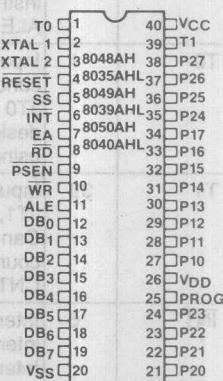


Figure 3.  
Pin Configuration



Table 1. Pin Description

Symbol	Pin No.	Function	Symbol	Pin No.	Function
V <sub>SS</sub>	20	Circuit GND potential	RD	8	Output strobe activated during a BUS read. Can be used to enable data onto the bus from an external device.
V <sub>DD</sub>	26	+5V during normal operation. Low power standby pin.			Used as a read strobe to external data memory. (Active low)
V <sub>CC</sub>	40	Main power supply; +5V during operation.	RESET	4	Input which is used to initialize the processor. (Active low) (Non TTL V <sub>IH</sub> )
PROG	25	Output strobe for 8243 I/O expander.			Used during power down.
P10-P17 Port 1	27-34	8-bit quasi-bidirectional port.			Used during ROM verification.
P20-P23 P24-P27 Port 2	21-24 35-38	8-bit quasi-bidirectional port. P20-P23 contain the four high order program counter bits during an external program memory fetch and serve as a 4-bit I/O expander bus for 8243.	WR	10	Output strobe during a bus write. (Active low) Used as write strobe to external data memory.
DB0-DB7 BUS	12-19	True bidirectional port which can be written or read synchronously using the RD, WR strobes. The port can also be statically latched.  Contains the 8 low order program counter bits during an external program memory fetch, and receives the addressed instruction under the control of PSEN. Also contains the address and data during an external RAM data store instruction, under control of ALE, RD, and WR.	ALE	11	Address latch enable. This signal occurs once during each cycle and is useful as a clock output.  The negative edge of ALE strobes address into external data and program memory.
T0	1	Input pin testable using the conditional transfer instructions JT0 and JNT0. T0 can be designated as a clock output using ENT0 CLK instruction	PSEN	9	Program store enable. This output occurs only during a fetch to external program memory. (Active low)
T1	39	Input pin testable using the JT1, and JNT1 instructions. Can be designated the timer/counter input using the STRT CNT instruction.	SS	5	Single step input can be used in conjunction with ALE to "single step" the processor through each instruction. (Active Low) Used in sync mode
INT	6	Interrupt input. Initiates an interrupt if interrupt is enabled. Interrupt is disabled after a reset. Also testable with conditional jump instruction. (Active low) interrupt must remain low for at least 3 machine cycles for proper operation.	EA	7	External access input which forces all program memory fetches to reference external memory. Useful for emulation and debug. (Active high) Used during ROM verification (12V)
			XTAL1	2	One side of crystal input for internal oscillator. Also input for external source. (Non TTL V <sub>IH</sub> )
			XTAL2	3	Other side of crystal input.

Table 2. Instruction Set

Accumulator			
Mnemonic	Description	Bytes	Cycles
ADD A, R	Add register to A	1	1
ADD A, @R	Add data memory to A	1	1
ADD A, # data	Add immediate to A	2	2
ADDC A, R	Add register with carry	1	1
ADDC A, @R	Add data memory with carry	1	1
ADDC A, # data	Add immediate with carry	2	2
ANL A, R	And register to A	1	1
ANL A, @R	And data memory to A	1	1
ANL A, # data	And immediate to A	2	2
ORL A, R	Or register to A	1	1
ORL A, @R	Or data memory to A	1	1
ORL A, # data	Or immediate to A	2	2
XRL A, R	Exclusive or register to A	1	1
XRL A, @R	Exclusive or data memory to A	1	1
XRL A, # data	Exclusive or immediate to A	2	2
INC A	Increment A	1	1
DEC A	Decrement A	1	1
CLR A	Clear A	1	1
CPL A	Complement A	1	1
DA A	Decimal adjust A	1	1
SWAP A	Swap nibbles of A	1	1
RL A	Rotate A left	1	1
RLC A	Rotate A left through carry	1	1
RR A	Rotate A right	1	1
RRC A	Rotate A right through carry	1	1

Input/Output			
Mnemonic	Description	Bytes	Cycles
IN A, P	Input port to A	1	2
OUTL P, A	Output A to port	1	2
ANL P, # data	And immediate to port	2	2
ORL P, # data	Or immediate to port	2	2
INS A, BUS	Input BUS to A	1	2
OUTL BUS, A	Output A to BUS	1	2
ANL BUS, # data	And immediate to BUS	2	2
ORL BUS, # data	Or immediate to BUS	2	2
MOVD A, P	Input expander port to A	1	2
MOVD P, A	Output A to expander port	1	2
ANLD P, A	And A to expander port	1	2
ORLD P, A	Or A to expander port	1	2

Registers			
Mnemonic	Description	Bytes	Cycles
INC R	Increment register	1	1
INC @R	Increment data memory	1	1
DEC R	Decrement register	1	1

Branch			
Mnemonic	Description	Bytes	Cycles
JMP addr	Jump unconditional	2	2
JMPP @A	Jump indirect	1	2
DJNZ R, addr	Decrement register and skip	2	2
JC addr	Jump on carry = 1	2	2
JNC addr	Jump on carry = 0	2	2
JZ addr	Jump on A zero	2	2
JNZ addr	Jump on A not zero	2	2
JT0 addr	Jump on T0 = 1	2	2
JNT0 addr	Jump on T0 = 0	2	2
JT1 addr	Jump on T1 = 1	2	2
JNT1 addr	Jump on T1 = 0	2	2
JF0 addr	Jump on F0 = 1	2	2
JF1 addr	Jump on F1 = 1	2	2
JTF addr	Jump on timer flag	2	2
JNI addr	Jump on INT = 0	2	2
JBb addr	Jump on accumulator bit	2	2

Subroutine			
Mnemonic	Description	Bytes	Cycles
CALL addr	Jump to subroutine	2	2
RET	Return	1	2
RETR	Return and restore status	1	2

Flags			
Mnemonic	Description	Bytes	Cycles
CLR C	Clear carry	1	1
CPL C	Complement carry	1	1
CLR F0	Clear flag 0	1	1
CPL F0	Complement flag 0	1	1
CLR F1	Clear flag 1	1	1
CPL F1	Complement flag 1	1	1

Table 2. Instruction Set (Continued)

Data Moves			
Mnemonic	Description	Bytes	Cycles
MOV A, R	Move register to A	1	1
MOV A, @R	Move data memory to A	1	1
MOV A, # data	Move immediate to A	2	2
MOV R, A	Move A to register	1	1
MOV @R, A	Move A to data memory	1	1
MOV R, # data	Move immediate to register	2	2
MOV @R, # data	Move immediate to data memory	2	2
MOV A, PSW	Move PSW to A	1	1
MOV PSW, A	Move A to PSW	1	1
XCH A, R	Exchange A and register	1	1
XCH A, @R	Exchange A and data memory	1	1
XCHD A, @R	Exchange nibble of A and register	1	1
MOVX A, @R	Move external data memory to A	1	2
MOVX @R, A	Move A to external data memory	1	2
MOVP A, @A	Move to A from current page	1	2
MOVP3 A, @	Move to A from page 3	1	2

Timer/Counter			
Mnemonic	Description	Bytes	Cycles
MOV A, T	Read timer/counter	1	1
MOV T, A	Load timer/counter	1	1
STRT T	Start timer	1	1
STRT CNT	Start timer	1	1
STOP TCNT	Stop timer/counter	1	1
EN TCNTI	Enable timer/counter interrupt	1	1
DIS TCNTI	Disable timer/counter interrupt	1	1

Control			
Mnemonic	Description	Bytes	Cycles
EN I	Enable external interrupt	1	1
DIS I	Disable external interrupt	1	1
SEL RB0	Select register bank 0	1	1
SEL RB1	Select register bank 1	1	1
SEL MB0	Select memory bank 0	1	1
SEL MB1	Select memory bank 1	1	1
ENT0 CLK	Enable clock output on T0	1	1

Mnemonic	Description	Bytes	Cycles
NOP	No operation	1	1
IDL	Select Idle Operation	1	1

Mnemonic	Description	Bytes	Cycles
CALL	Jump to subroutine	2	2
RET	Return	1	2
RETR	Return and restore status	1	2

Mnemonic	Description	Bytes	Cycles
CLR C	Clear carry	1	1
CPL C	Complement carry	1	1
CLR F0	Clear flag 0	1	1
CPL F0	Complement flag 0	1	1
CLR F1	Clear flag 1	1	1
CPL F1	Complement flag 1	1	1

Mnemonic	Description	Bytes	Cycles
IN A, P	Input port to A	1	2
OUT A, P	Output A to port	1	2
ANL P, # data	And immediate to port	2	2
ORL P, # data	Or immediate to port	2	2
IN A, BUS	Input BUS to A	1	2
OUT A, BUS	Output A to BUS	1	2
ANL BUS, # data	And immediate to BUS	2	2
ORL BUS, # data	Or immediate to BUS	2	2
MOV A, R	Input expander port to A	1	2
MOV R, A	Output A to expander port	1	2
ANL P, A	And A to expander port	1	2
ORL P, A	Or A to expander port	1	2

# **ABSOLUTE MAXIMUM RATINGS\***

Ambient Temperature Under Bias . . . 0°C to 70°C  
Storage Temperature . . . . . -65°C to +150°C  
Voltage On Any Pin With Respect  
to Ground . . . . . -0.5V to +7V  
Power Dissipation . . . . . 1.5 Watt

\*NOTICE: Stresses above those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of device at these or any other conditions above those indicated in the operational sections of this specification is not implied.

## **D.C. CHARACTERISTICS:** (T<sub>A</sub> = 0°C to 70°C; V<sub>CC</sub> = V<sub>DD</sub> = 5V ± 10%; V<sub>SS</sub> = 0V)

Symbol	Parameter	Limits			Unit	Test Conditions	Device
		Min	Typ	Max			
V <sub>IL</sub>	Input Low Voltage (All Except RESET, X1, X2)	-.5		.8	V		All
V <sub>IL1</sub>	Input Low Voltage (RESET, X1, X2)	-.5		.6	V		All
V <sub>IH</sub>	Input High Voltage (All Except XTAL1, XTAL2, RESET)	2.0		V <sub>CC</sub>	V		All
V <sub>IH1</sub>	Input High Voltage (X1, X2, RESET)	3.8		V <sub>CC</sub>	V		All
V <sub>OL</sub>	Output Low Voltage (BUS)			.45	V	I <sub>OL</sub> = 2.0 mA	All
V <sub>OL1</sub>	Output Low Voltage (RD, WR, PSEN, ALE)			.45	V	I <sub>OL</sub> = 1.8 mA	All
V <sub>OL2</sub>	Output Low Voltage (PROG)			.45	V	I <sub>OL</sub> = 1.0 mA	All
V <sub>OL3</sub>	Output Low Voltage (All Other Outputs)			.45	V	I <sub>OL</sub> = 1.6 mA	All
V <sub>OH</sub>	Output High Voltage (BUS)	2.4			V	I <sub>OH</sub> = -400 μA	All
V <sub>OH1</sub>	Output High Voltage (RD, WR, PSEN, ALE)	2.4			V	I <sub>OH</sub> = -100 μA	All
V <sub>OH2</sub>	Output High Voltage (All Other Outputs)	2.4			V	I <sub>OH</sub> = -40 μA	All



Symbol	Parameter	Limits			Unit	Test Conditions	Device
		Min	Typ	Max			
I <sub>L1</sub>	Leakage Current (T1, INT)			± 10	μA	V <sub>SS</sub> ≤ V <sub>IN</sub> ≤ V <sub>CC</sub>	All
I <sub>LI1</sub>	Input Leakage Current (P10-P17, P20-P27, EA, SS)			- 500	μA	V <sub>SS</sub> + .45 ≤ V <sub>IN</sub> ≤ V <sub>CC</sub>	All
I <sub>LI2</sub>	Input Leakage Current RESET	20		300	μA	V <sub>SS</sub> ≤ V <sub>IN</sub> ≤ 3.8V	All
I <sub>L0</sub>	Leakage Current (BUS, T0) (High Impedance State)			± 10	μA	V <sub>SS</sub> ≤ V <sub>IN</sub> ≤ V <sub>CC</sub>	All
I <sub>DD</sub>	V <sub>DD</sub> Supply Current (RAM Standby)		3	5	mA		8048AH 8035AHL
			4	7	mA		8049AH 8039AHL
			5	10	mA		8050AH 8040AHL
I <sub>DD</sub> + I <sub>CC</sub>	Total Supply Current*		30	65	mA		8048AH 8035AHL
			35	70	mA		8049AH 8039AHL
			40	80	mA		8050AH 8040AHL
V <sub>DD</sub>	RAM Standby Voltage	2.2		5.5	V	Standby Mode Reset ≤ V <sub>IL1</sub>	All

\*I<sub>CC</sub> + I<sub>DD</sub> is measured with all outputs disconnected;  $\overline{\text{SS}}$ ,  $\overline{\text{RESET}}$ , and  $\overline{\text{INT}}$  equal to V<sub>CC</sub>; EA equal to V<sub>SS</sub>.

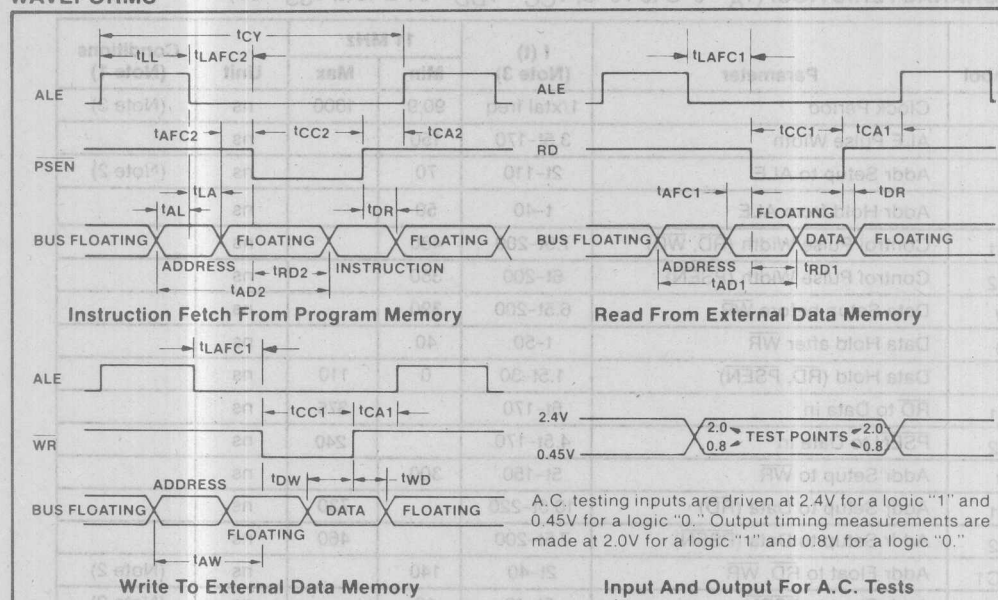
**A.C. CHARACTERISTICS:** ( $T_A = 0^\circ\text{C}$  to  $70^\circ\text{C}$ ;  $V_{CC} = V_{DD} = 5V \pm 10\%$ ;  $V_{SS} = 0V$ )

Symbol	Parameter	f (t) (Note 3)	11 MHz		Unit	Conditions (Note 1)
			Min	Max		
t	Clock Period	1/xtal freq	90.9	1000	ns	(Note 3)
t <sub>LL</sub>	ALE Pulse Width	3.5t-170	150		ns	
t <sub>AL</sub>	Addr Setup to ALE	2t-110	70		ns	(Note 2)
t <sub>LA</sub>	Addr Hold from ALE	t-40	50		ns	
t <sub>CC1</sub>	Control Pulse Width ( $\overline{RD}$ , $\overline{WR}$ )	7.5t-200	480		ns	
t <sub>CC2</sub>	Control Pulse Width ( $\overline{PSEN}$ )	6t-200	350		ns	
t <sub>DW</sub>	Data Setup before $\overline{WR}$	6.5t-200	390		ns	
t <sub>WD</sub>	Data Hold after $\overline{WR}$	t-50	40		ns	
t <sub>DR</sub>	Data Hold ( $\overline{RD}$ , $\overline{PSEN}$ )	1.5t-30	0	110	ns	
t <sub>RD1</sub>	$\overline{RD}$ to Data in	6t-170		375	ns	
t <sub>RD2</sub>	$\overline{PSEN}$ to Data in	4.5t-170		240	ns	
t <sub>AW</sub>	Addr Setup to $\overline{WR}$	5t-150	300		ns	
t <sub>AD1</sub>	Addr Setup to Data ( $\overline{RD}$ )	10.5t-220		730	ns	
t <sub>AD2</sub>	Addr Setup to Data ( $\overline{PSEN}$ )	7.5t-200		460	ns	
t <sub>AFC1</sub>	Addr Float to $\overline{RD}$ , $\overline{WR}$	2t-40	140		ns	(Note 2)
t <sub>AFC2</sub>	Addr Float to $\overline{PSEN}$	.5t-40	10		ns	(Note 2)
t <sub>LAFC1</sub>	ALE to Control ( $\overline{RD}$ , $\overline{WR}$ )	3t-75	200		ns	
t <sub>LAFC2</sub>	ALE to Control ( $\overline{PSEN}$ )	1.5t-75	60		ns	
t <sub>CA1</sub>	Control to ALE ( $\overline{RD}$ , $\overline{WR}$ , $\overline{PROG}$ )	t-65	25		ns	
t <sub>CA2</sub>	Control to ALE ( $\overline{PSEN}$ )	4t-70	290		ns	
t <sub>CP</sub>	Port Control Setup to $\overline{PROG}$	1.5t-80	50		ns	
t <sub>PC</sub>	Port Control Hold to $\overline{PROG}$	4t-260	100		ns	
t <sub>PR</sub>	$\overline{PROG}$ to P2 Input Valid	8.5t-120		650	ns	
t <sub>PF</sub>	Input Data Hold from $\overline{PROG}$	1.5t	0	140	ns	
t <sub>DP</sub>	Output Data Setup	6t-290	250		ns	
t <sub>PD</sub>	Output Data Hold	1.5t-90	40		ns	
t <sub>PP</sub>	$\overline{PROG}$ Pulse Width	10.5t-250	700		ns	
t <sub>PL</sub>	Port 2 I/O Setup to ALE	4t-200	160		ns	
t <sub>LP</sub>	Port 2 I/O Hold to ALE	.5t-30	15		ns	
t <sub>PV</sub>	Port Output from ALE	4.5t+100		510	ns	
t <sub>OPRR</sub>	T0 Rep Rate	3t	270		ns	
t <sub>CY</sub>	Cycle Time	15t	1.36	15.0	$\mu\text{s}$	

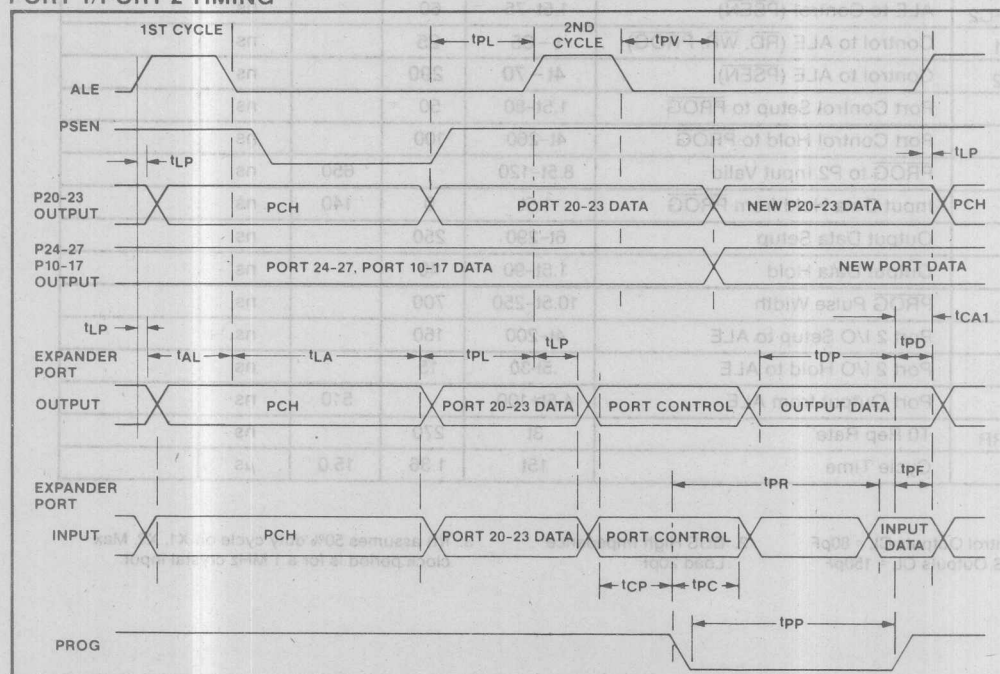
**Notes:**

- Control Outputs  $CL = 80\text{pF}$   
BUS Outputs  $CL = 150\text{pF}$
- BUS High Impedance  
Load  $20\text{pF}$
- f(t) assumes 50% duty cycle on X1, X2. Max clock period is for a 1 MHz crystal input.

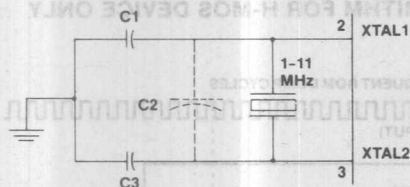
## WAVEFORMS



### PORT 1/PORT 2 TIMING



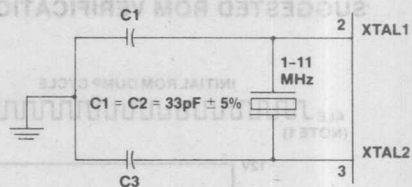
### CRYSTAL OSCILLATOR MODE



$$\begin{aligned} C1 &= 5\text{pF} \pm 1/2\text{pF} + (\text{STRAY} < 5\text{pF}) \\ C2 &= (\text{CRYSTAL} + \text{STRAY}) < 8\text{pF} \\ C3 &= 20\text{pF} \pm 1\text{pF} + (\text{STRAY} < 5\text{pF}) \end{aligned}$$

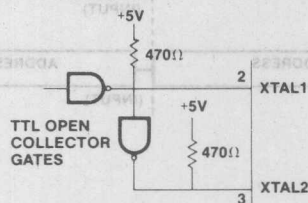
Crystal series resistance should be less than 30Ω at 11 MHz;  
less than 75Ω at 6 MHz; less than 180Ω at 3.6 MHz.

### CERAMIC RESONATOR MODE



$$C1 = C2 = 33\text{pF} \pm 5\%$$

### DRIVING FROM EXTERNAL SOURCE

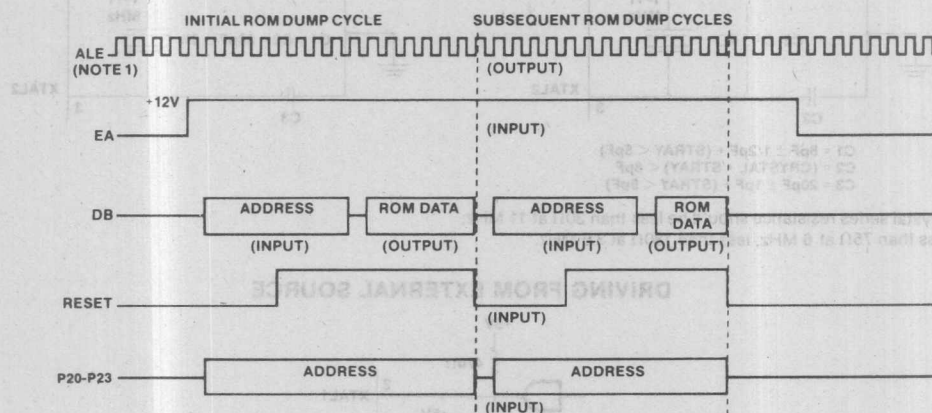


For XTAL1 and XTAL2 define "high" as voltages above 1.6V  
and "low" as voltages below 1.6V. The duty cycle require-  
ments for externally driving XTAL1 and XTAL2 using the

circuit shown above are as follows: XTAL1 must be high 35-  
65% of the period and XTAL2 must be high 36-65% of the  
period. Rise and fall times must be faster than 20 nS.



# SUGGESTED ROM VERIFICATION ALGORITHM FOR H-MOS DEVICE ONLY



VCC = VDD = + 5V  
VSS = 0V

	48H	49H	50H
A10	0	ADDR	ADDR
A11	0	0	ADDR

**NOTE:** ALE is function of X1, X2 inputs.

# 8748H/8035H/8749H/8039H HMOS-E SINGLE-COMPONENT 8-BIT MICROCOMPUTER

- High Performance HMOS-E
- Interval Timer/Event Counter
- Two Single Level Interrupts
- Single 5-Volt Supply
- Over 96 Instructions, 90% Single Byte
- Compatible with 8080/8085 Peripherals
- Easily Expandable Memory and I/O
- Up to 1.35  $\mu$ Sec Instruction Cycle
- All Instructions 1 or 2 cycles

The Intel 8749H/8039H/8748H/8035H are totally self-sufficient, 8-bit parallel computers fabricated on single silicon chips using Intel's advanced N-channel silicon gate HMOS-E process.

The family contains 27 I/O lines, an 8-bit timer/counter, on-chip RAM and on-board oscillator/clock circuits. For systems that require extra capability, the family can be expanded using MCS<sup>®</sup>-80/MCS<sup>®</sup>-85 peripherals.

These microcomputers are designed to be efficient controllers as well as arithmetic processors. They have extensive bit handling capability as well as facilities for both binary and BCD arithmetic. Efficient use of program memory results from an instruction set consisting mostly of single byte instructions and no instructions over 2 bytes in length.

Device	Internal Memory
8039H	none 128 x 8 RAM
8035H	none 64 x 8 RAM
8749H	2K x 8 EPROM 128 x 8 RAM
8748H	1K x 8 EPROM 64 x 8 RAM

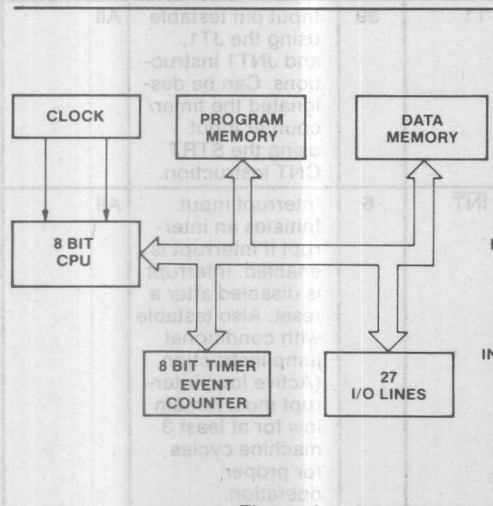


Figure 1.  
Block Diagram

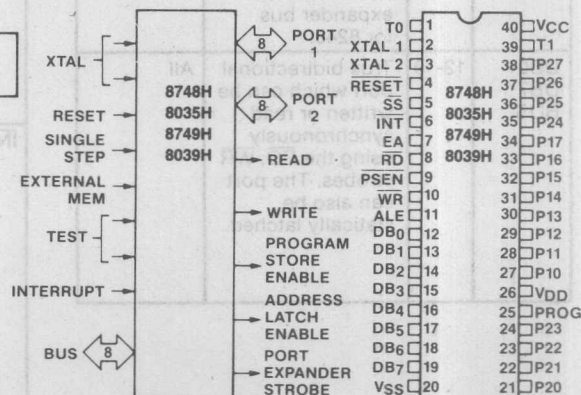


Figure 2.  
Logic Symbol

Figure 3.  
Pin Configuration

### Table 1. Pin Description

Symbol	Pin No.	Function	Device
V <sub>SS</sub>	20	Circuit GND potential	All
V <sub>DD</sub>	26	+5V during normal operation.	All
		Programming power supply (+21V).	8748H 8749H
V <sub>CC</sub>	40	Main power supply; +5V during operation and programming.	All
PROG	25	Output strobe for 8243 I/O expander.	All
		Program pulse (+18V) input pin during programming.	8748H 8749H (See Note)
P10-P17 Port 1	27-34	8-bit quasi-bidirectional port.	All
P20-P23 P24-P27 Port 2	21-24 35-38	8-bit quasi-bidirectional port. P20-P23 contain the four high order program counter bits during an external program memory fetch and serve as a 4-bit I/O expander bus for 8243.	All
DB0-DB7 BUS	12-19	True bidirectional port which can be written or read synchronously using the RD, WR strobes. The port can also be statically latched.	All

Symbol	Pin No.	Function	Device
(Con't)		Contains the 8 low order program counter bits during an external program memory fetch, and receives the addressed instruction under the control of PSEN. Also contains the address and data during an external RAM data store instruction, under control of ALE, RD, and WR.	
T0	1	Input pin testable using the conditional transfer instructions JT0 and JNT0. T0 can be designated as a clock output using ENT0 CLK instruction	All
		Used during programming.	8748H 8749H
T1	39	Input pin testable using the JT1, and JNT1 instructions. Can be designated the timer/counter input using the STRT CNT instruction.	All
INT	6	Interrupt input. Initiates an interrupt if interrupt is enabled. Interrupt is disabled after a reset. Also testable with conditional jump instruction. (Active low) interrupt must remain low for at least 3 machine cycles for proper operation.	All

Table 1. Pin Description (Continued)

Symbol	Pin No.	Function	Device	Symbol	Pin No.	Function	Device
RD	8	Output strobe activated during a BUS read. Can be used to enable data onto the bus from an external device.  Used as a read strobe to external data memory. (Active low)	All	PSEN	9	Program store enable. This output occurs only during a fetch to external program memory. (Active low)	All
RESET	4	Input which is used to initialize the processor. (Active low) (Non TTL $V_{IH}$ )  Used during programming.	All 8748H 8749H	SS	5	Single step input can be used in conjunction with ALE to "single step" the processor through each instruction.	All
WR	10	Output strobe during a bus write. (Active low)  Used as write strobe to external data memory.	All	EA	7	External access input which forces all program memory fetches to reference external memory. Useful for emulation and debug. (Active high)  Used during (18V) programming	All 8748H 8749H
ALE	11	Address latch enable. This signal occurs once during each cycle and is useful as a clock output.  The negative edge of ALE strobes address into external data and program memory.	All	XTAL1	2	One side of crystal input for internal oscillator. Also input for external source. (Non TTL $V_{IH}$ )	All
				XTAL2	3	Other side of crystal input.	All

**NOTE:** On the 8749H/8039H, PROG must be clamped to  $V_{CC}$  when not programming. A diode should be used when using an 8243; otherwise, a direct connection is permissible.



Table 2. Instruction Set

Accumulator			
Mnemonic	Description	Bytes	Cycles
ADD A, R	Add register to A	1	1
ADD A, @R	Add data memory to A	1	1
ADD A, # data	Add immediate to A	2	2
ADDC A, R	Add register with carry	1	1
ADDC A, @R	Add data memory with carry	1	1
ADDC A, # data	Add immediate with carry	2	2
ANL A, R	And register to A	1	1
ANL A, @R	And data memory to A	1	1
ANL A, # data	And immediate to A	2	2
ORL A, R	Or register to A	1	1
ORL A, @R	Or data memory to A	1	1
ORL A, # data	Or immediate to A	2	2
XRL A, R	Exclusive or register to A	1	1
XRL A, @R	Exclusive or data memory to A	1	1
XRL A, # data	Exclusive or immediate to A	2	2
INC A	Increment A	1	1
DEC A	Decrement A	1	1
CLR A	Clear A	1	1
CPL A	Complement A	1	1
DA A	Decimal adjust A	1	1
SWAP A	Swap nibbles of A	1	1
RL A	Rotate A left	1	1
RLC A	Rotate A left through carry	1	1
RR A	Rotate A right	1	1
RRC A	Rotate A right through carry	1	1

Input/Output			
Mnemonic	Description	Bytes	Cycles
IN A, P	Input port to A	1	2
OUTL P, A	Output A to port	1	2
ANL P, # data	And immediate to port	2	2
ORL P, # data	Or immediate to port	2	2
INS A, BUS	Input BUS to A	1	2
OUTL BUS, A	Output A to BUS	1	2
ANL BUS, # data	And immediate to BUS	2	2
ORL BUS, # data	Or immediate to BUS	2	2
MOVD A, P	Input expander port to A	1	2
MOVD P, A	Output A to expander port	1	2
ANLD P, A	And A to expander port	1	2
ORLD P, A	Or A to expander port	1	2

Registers			
Mnemonic	Description	Bytes	Cycles
INC R	Increment register	1	1
INC @R	Increment data memory	1	1
DEC R	Decrement register	1	1

Branch			
Mnemonic	Description	Bytes	Cycles
JMP addr	Jump unconditional	2	2
JMPP @A	Jump indirect	1	2
DJNZ R, addr	Decrement register and skip	2	2
JC addr	Jump on carry = 1	2	2
JNC addr	Jump on carry = 0	2	2
JZ addr	Jump on A zero	2	2
JNZ addr	Jump on A not zero	2	2
JT0 addr	Jump on T0 = 1	2	2
JNT0 addr	Jump on T0 = 0	2	2
JT1 addr	Jump on T1 = 1	2	2
JNT1 addr	Jump on T1 = 0	2	2
JF0 addr	Jump on F0 = 1	2	2
JF1 addr	Jump on F1 = 1	2	2
JTF addr	Jump on timer flag	2	2
JNI addr	Jump on INT = 0	2	2
JBb addr	Jump on accumulator bit	2	2

Subroutine			
Mnemonic	Description	Bytes	Cycles
CALL addr	Jump to subroutine	2	2
RET	Return	1	2
RETR	Return and restore status	1	2

Flags			
Mnemonic	Description	Bytes	Cycles
CLR C	Clear carry	1	1
CPL C	Complement carry	1	1
CLR F0	Clear flag 0	1	1
CPL F0	Complement flag 0	1	1
CLR F1	Clear flag 1	1	1
CPL F1	Complement flag 1	1	1

Table 2. Instruction Set (Continued)

Data Moves				Timer/Counter			
Mnemonic	Description	Bytes	Cycles	Mnemonic	Description	Bytes	Cycles
MOV A, R	Move register to A	1	1	MOV A, T	Read timer/counter	1	1
MOV A, @R	Move data memory to A	1	1	MOV T, A	Load timer/counter	1	1
MOV A, # data	Move immediate to A	2	2	STRT T	Start timer	1	1
MOV R, A	Move A to register	1	1	STRT CNT	Start counter	1	1
MOV @R, A	Move A to data memory	1	1	STOP TCNT	Stop timer/counter	1	1
MOV R, # data	Move immediate to register	2	2	EN TCNTI	Enable timer/counter interrupt	1	1
MOV @R, # data	Move immediate to data memory	2	2	DIS TCNTI	Disable timer/counter interrupt	1	1
MOV A, PSW	Move PSW to A	1	1	Control			
MOV PSW, A	Move A to PSW	1	1	Mnemonic	Description	Bytes	Cycles
XCH A, R	Exchange A and register	1	1	EN I	Enable external interrupt	1	1
XCH A, @R	Exchange A and data memory	1	1	DIS I	Disable external interrupt	1	1
XCHD A, @R	Exchange nibble of A and register	1	1	SEL RB0	Select register bank 0	1	1
MOVX A, @R	Move external data memory to A	1	2	SEL RB1	Select register bank 1	1	1
MOVX @R, A	Move A to external data memory	1	2	SEL MB0	Select memory bank 0	1	1
MOVP A, @A	Move to A from current page	1	2	SEL MB1	Select memory bank 1	1	1
MOVP3 A, @A	Move to A from page 3	1	2	ENT0 CLK	Enable clock output on T0	1	1
				Mnemonic	Description	Bytes	Cycles
				NOP	No operation	1	1

**ABSOLUTE MAXIMUM RATINGS\***

Ambient Temperature Under Bias . . . 0°C to 70°C  
 Storage Temperature . . . . . -65°C to +150°C  
 Voltage On Any Pin With Respect  
   to Ground . . . . . -0.5V to +7V  
 Power Dissipation . . . . . 1.0 Watt

\*NOTICE: Stresses above those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of device at these or any other conditions above those indicated in the operational sections of this specification is not implied.

**D.C. CHARACTERISTICS:** (TA = 0°C to 70°C; VCC = VDD = 5V ± 10%; VSS = 0V)

Symbol	Parameter	Limits			Unit	Test Conditions	Device
		Min	Typ	Max			
V <sub>IL</sub>	Input Low Voltage (All Except RESET, X1, X2)	- .5		.8	V		All
V <sub>IL1</sub>	Input Low Voltage (RESET, X1, X2)	- .5		.6	V		All
V <sub>IH</sub>	Input High Voltage (All Except XTAL1, XTAL2, RESET)	2.0		V <sub>CC</sub>	V		All
V <sub>IH1</sub>	Input High Voltage (X1, X2, RESET)	3.8		V <sub>CC</sub>	V		All
V <sub>OL</sub>	Output Low Voltage (BUS)			.45	V	I <sub>OL</sub> = 2.0 mA	All
V <sub>OL1</sub>	Output Low Voltage (RD, WR, PSEN, ALE)			.45	V	I <sub>OL</sub> = 1.8 mA	All
V <sub>OL2</sub>	Output Low Voltage (PROG)			.45	V	I <sub>OL</sub> = 1.0 mA	All
V <sub>OL3</sub>	Output Low Voltage (All Other Outputs)			.45	V	I <sub>OL</sub> = 1.6 mA	All
V <sub>OH</sub>	Output High Voltage (BUS)	2.4			V	I <sub>OH</sub> = -400 μA	All
V <sub>OH1</sub>	Output High Voltage (RD, WR, PSEN, ALE)	2.4			V	I <sub>OH</sub> = -100 μA	All
V <sub>OH2</sub>	Output High Voltage (All Other Outputs)	2.4			V	I <sub>OH</sub> = -40 μA	All

**D.C. CHARACTERISTICS:** ( $T_A = 0^\circ\text{C}$  to  $70^\circ\text{C}$ ;  $V_{CC} = V_{DD} = 5V \pm 10\%$ ;  $V_{SS} = 0V$ ) (Continued)

Symbol	Parameter	Limits			Unit	Test Conditions	Device
		Min	Typ	Max			
$I_{L1}$	Leakage Current ( $T_1$ , INT)			$\pm 10$	$\mu\text{A}$	$V_{SS} \leq V_{IN} \leq V_{CC}$	All
$I_{LI1}$	Input Leakage Current (P10–P17, P20–P27, EA, SS)			– 500	$\mu\text{A}$	$V_{SS} + .45 \leq V_{IN} \leq V_{CC}$	All
$I_{LI2}$	Input Leakage Current RESET	– 10		– 300	$\mu\text{A}$	$V_{SS} \leq V_{IN} \leq 3.8V$	All
$I_{L0}$	Leakage Current (BUS, $T_0$ ) (High Impedance State)			$\pm 10$	$\mu\text{A}$	$V_{SS} \leq V_{IN} \leq V_{CC}$	All
$I_{DD} + I_{CC}$	Total Supply Current*		80	100	mA		8035H
			95	110	mA		8039H
			80	100	mA		8748H
			95	110	mA		8749H

\* $I_{CC} + I_{DD}$  is measured with all outputs disconnected;  $\overline{SS}$ , RESET, and INT equal to  $V_{CC}$ ; EA equal to  $V_{SS}$ .



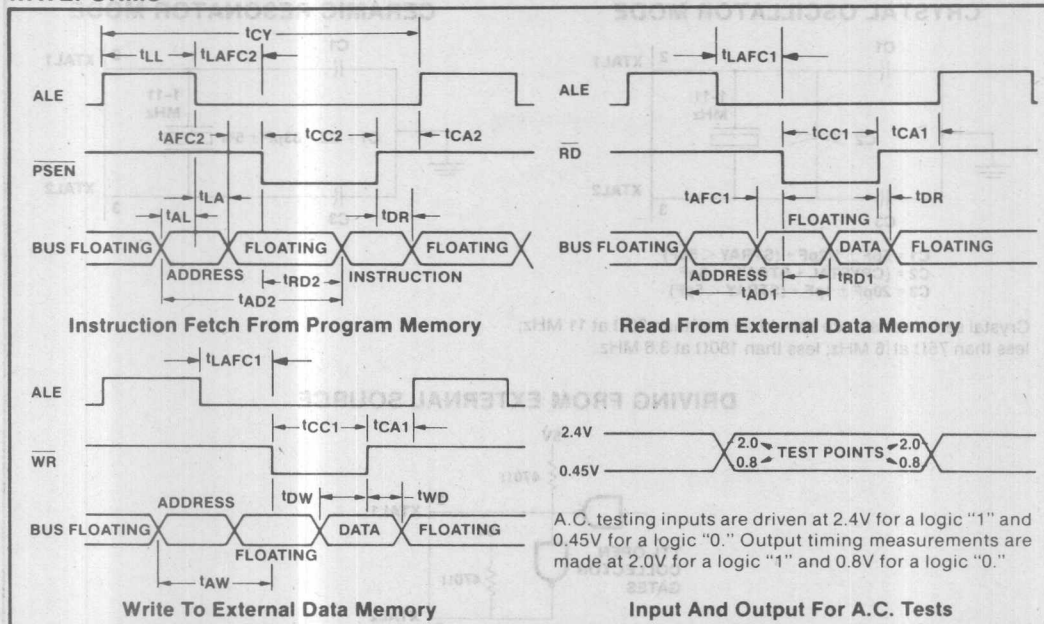
**A.C. CHARACTERISTICS:** ( $T_A = 0^\circ\text{C}$  to  $70^\circ\text{C}$ ;  $V_{CC} = V_{DD} = 5V \pm 10\%$ ;  $V_{SS} = 0V$ )

Symbol	Parameter	f (t) (Note 3)	11 MHz		Unit	Conditions (Note 1)
			Min	Max		
t	Clock Period	1/xtal freq	90.9	1000	ns	(Note 3)
t <sub>LL</sub>	ALE Pulse Width	3.5t-170	150		ns	
t <sub>AL</sub>	Addr Setup to ALE	2t-110	70		ns	(Note 2)
t <sub>LA</sub>	Addr Hold from ALE	t-40	50		ns	
t <sub>CC1</sub>	Control Pulse Width ( $\overline{RD}$ , $\overline{WR}$ )	7.5t-200	480		ns	
t <sub>CC2</sub>	Control Pulse Width ( $\overline{PSEN}$ )	6t-200	350		ns	
t <sub>DW</sub>	Data Setup before $\overline{WR}$	6.5t-200	390		ns	
t <sub>WD</sub>	Data Hold after $\overline{WR}$	t-50	40		ns	
t <sub>DR</sub>	Data Hold ( $\overline{RD}$ , $\overline{PSEN}$ )	1.5t-30	0	110	ns	
t <sub>RD1</sub>	$\overline{RD}$ to Data in	6t-170		375	ns	
t <sub>RD2</sub>	$\overline{PSEN}$ to Data in	4.5t-170		240	ns	
t <sub>AW</sub>	Addr Setup to $\overline{WR}$	5t-150	300		ns	
t <sub>AD1</sub>	Addr Setup to Data ( $\overline{RD}$ )	10.5t-220		730	ns	
t <sub>AD2</sub>	Addr Setup to Data ( $\overline{PSEN}$ )	7.5t-200		460	ns	
t <sub>AFC1</sub>	Addr Float to $\overline{RD}$ , $\overline{WR}$	2t-40	140		ns	(Note 2)
t <sub>AFC2</sub>	Addr Float to $\overline{PSEN}$	.5t-40	10		ns	(Note 2)
t <sub>LAFC1</sub>	ALE to Control ( $\overline{RD}$ , $\overline{WR}$ )	3t-75	200		ns	
t <sub>LAFC2</sub>	ALE to Control ( $\overline{PSEN}$ )	1.5t-75	60		ns	
t <sub>CA1</sub>	Control to ALE ( $\overline{RD}$ , $\overline{WR}$ , $\overline{PROG}$ )	t-65	25		ns	
t <sub>CA2</sub>	Control to ALE ( $\overline{PSEN}$ )	4t-70	290		ns	
t <sub>CP</sub>	Port Control Setup to $\overline{PROG}$	1.5t-80	50		ns	
t <sub>PC</sub>	Port Control Hold to $\overline{PROG}$	4t-260	100		ns	
t <sub>PR</sub>	$\overline{PROG}$ to P2 Input Valid	8.5t-120		650	ns	
t <sub>PF</sub>	Input Data Hold from $\overline{PROG}$	1.5t	0	140	ns	
t <sub>DP</sub>	Output Data Setup	6t-290	250		ns	
t <sub>PD</sub>	Output Data Hold	1.5t-90	40		ns	
t <sub>PP</sub>	$\overline{PROG}$ Pulse Width	10.5t-250	700		ns	
t <sub>PL</sub>	Port 2 I/O Setup to ALE	4t-200	160		ns	
t <sub>LP</sub>	Port 2 I/O Hold to ALE	.5t-30	15		ns	
t <sub>PV</sub>	Port Output from ALE	4.5t+100		510	ns	
t <sub>0PRR</sub>	T0 Rep Rate	3t	270		ns	
t <sub>CY</sub>	Cycle Time	15t	1.36	15.0	$\mu\text{s}$	

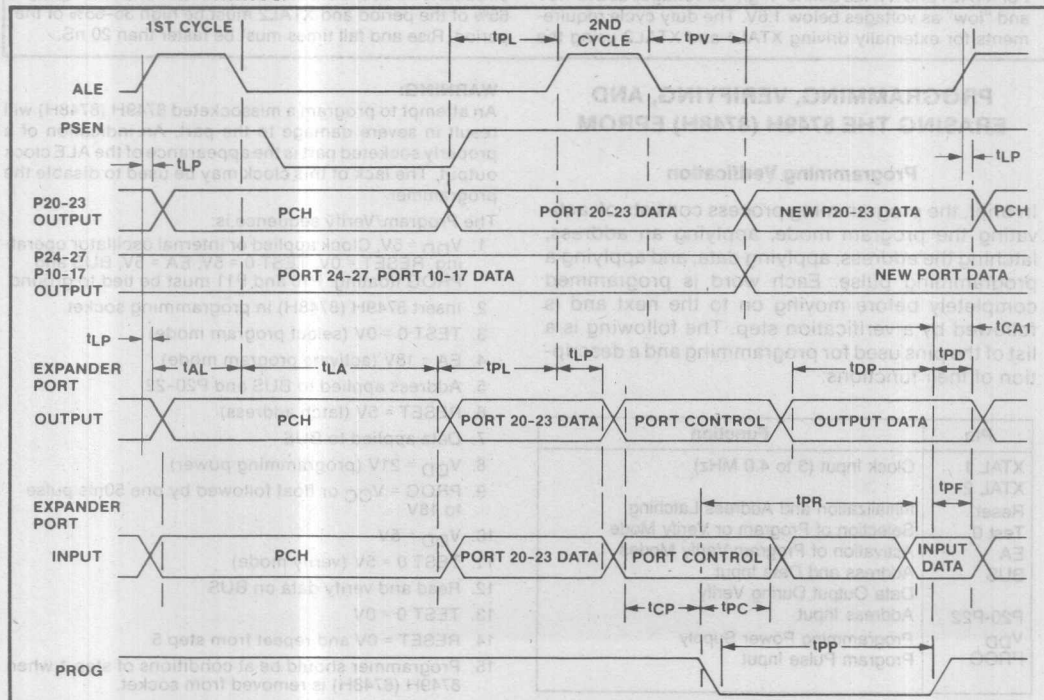
**Notes:**

- Control Outputs  $CL = 80\text{pF}$   
BUS Outputs  $CL = 150\text{pF}$
- BUS High Impedance  
Load  $20\text{pF}$
- f(t) assumes 50% duty cycle on X1, X2. Max clock period is for a 1 MHz crystal input.

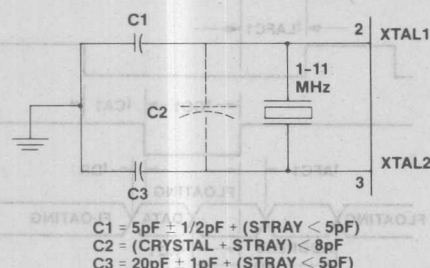
# WAVEFORMS



## PORT 1/PORT 2 TIMING

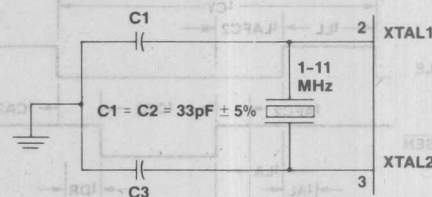


## CRYSTAL OSCILLATOR MODE

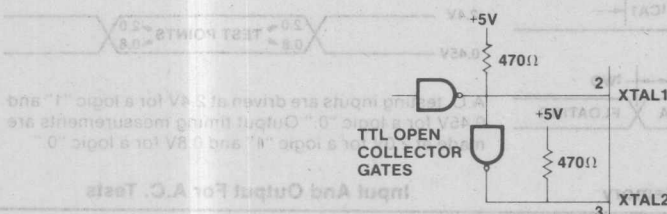


Crystal series resistance should be less than 30 $\Omega$  at 11 MHz;  
less than 75 $\Omega$  at 6 MHz; less than 180 $\Omega$  at 3.6 MHz.

## CERAMIC RESONATOR MODE



## DRIVING FROM EXTERNAL SOURCE



For XTAL1 and XTAL2 define "high" as voltages above 1.6V and "low" as voltages below 1.6V. The duty cycle requirements for externally driving XTAL1 and XTAL2 using the

circuit shown above are as follows: XTAL1 must be high 35–65% of the period and XTAL2 must be high 36–65% of the period. Rise and fall times must be faster than 20 nS.

## PROGRAMMING, VERIFYING, AND ERASING THE 8749H (8748H) EPROM

## Programming Verification

In brief, the programming process consists of: activating the program mode, applying an address, latching the address, applying data, and applying a programming pulse. Each word is programmed completely before moving on to the next and is followed by a verification step. The following is a list of the pins used for programming and a description of their functions:

Pin	Function
XTAL 1	Clock Input (3 to 4.0 MHz)
XTAL 2	
Reset	Initialization and Address Latching
Test 0	Selection of Program or Verify Mode
EA	Activation of Program/Verify Modes
BUS	Address and Data Input Data Output During Verify
P20-P22	Address Input
VDD	Programming Power Supply
PROG	Program Pulse Input

## WARNING:

An attempt to program a missocketed 8749H (8748H) will result in severe damage to the part. An indication of a properly socketed part is the appearance of the ALE clock output. The lack of this clock may be used to disable the programmer.

The Program/Verify sequence is:

1.  $V_{DD} = 5V$ , Clock applied or internal oscillator operating, RESET = 0V, TEST 0 = 5V, EA = 5V, BUS and PROG floating; P10 and P11 must be tied to ground.
2. Insert 8749H (8748H) in programming socket.
3. TEST 0 = 0V (select program mode)
4. EA = 18V (activate program mode)
5. Address applied to BUS and P20–22
6. RESET = 5V (latch address)
7. Data applied to BUS
8.  $V_{DD} = 21V$  (programming power)
9. PROG =  $V_{CC}$  or float followed by one 50ms pulse to 18V
10.  $V_{DD} = 5V$
11. TEST 0 = 5V (verify mode)
12. Read and verify data on BUS
13. TEST 0 = 0V
14. RESET = 0V and repeat from step 5
15. Programmer should be at conditions of step 1 when 8749H (8748H) is removed from socket.

**A.C. TIMING SPECIFICATION FOR PROGRAMMING 8748H/8749H ONLY:**

( $T_A = 25^\circ\text{C} \pm 5^\circ\text{C}$ ;  $V_{CC} = 5\text{V} \pm 5\%$ ;  $V_{DD} = 21 \pm .5\text{V}$ )

Symbol	Parameter	Min	Max	Unit	Test Conditions
$t_{AW}$	Address Setup Time to $\overline{\text{RESET}}$	$4t_{CY}$			
$t_{WA}$	Address Hold Time After $\overline{\text{RESET}}$	$4t_{CY}$			
$t_{DW}$	Data in Setup Time to $\text{PROG}$	$4t_{CY}$			
$t_{WD}$	Data in Hold Time After $\text{PROG}$	$4t_{CY}$			
$t_{PH}$	$\overline{\text{RESET}}$ Hold Time to Verify	$4t_{CY}$			
$t_{VDDW}$	$V_{DD}$ Hold Time Before $\text{PROG}$	0	1.0	ms	
$t_{VDDH}$	$V_{DD}$ Hold Time After $\text{PROG}$	0	1.0	ms	
$t_{PW}$	Program Pulse Width	50	60	ms	
$t_{TW}$	Test 0 Setup Time for Program Mode	$4t_{CY}$			
$t_{WT}$	Test 0 Hold Time After Program Mode	$4t_{CY}$			
$t_{DO}$	Test 0 to Data Out Delay		$4t_{CY}$		
$t_{WW}$	$\overline{\text{RESET}}$ Pulse Width to Latch Address	$4t_{CY}$			
$t_r, t_f$	$V_{DD}$ and $\text{PROG}$ Rise and Fall Times	0.5	100	$\mu\text{s}$	
$t_{CY}$	CPU Operation Cycle Time	3.75	5	$\mu\text{s}$	
$t_{RE}$	$\overline{\text{RESET}}$ Setup Time before $\text{EA}$	$4t_{CY}$			

**NOTE:** If Test 0 is high,  $t_{DO}$  can be triggered by  $\overline{\text{RESET}}$ .

**D.C. TIMING SPECIFICATION FOR PROGRAMMING 8748H/8749H ONLY:**

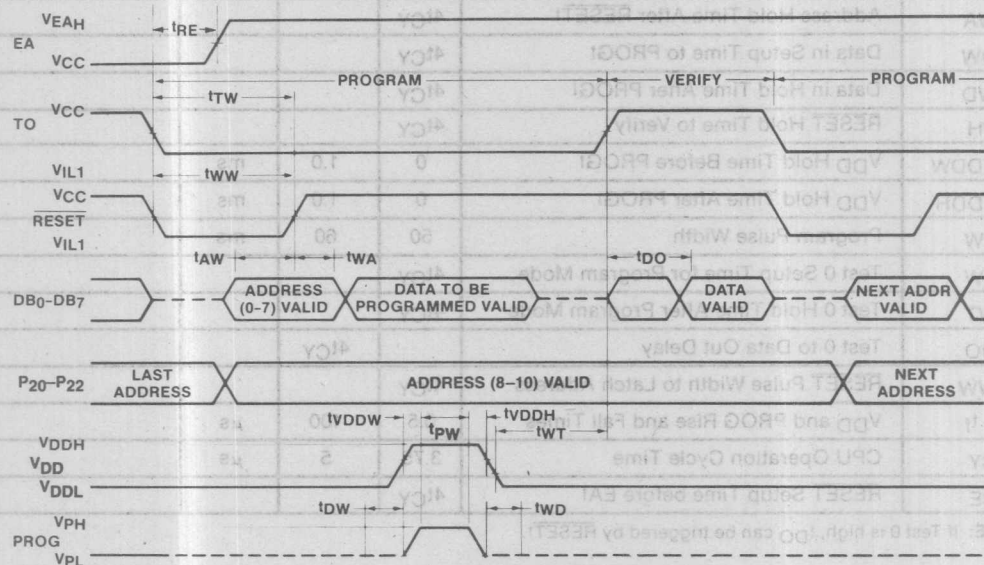
( $T_A = 25^\circ\text{C} \pm 5^\circ\text{C}$ ;  $V_{CC} = 5\text{V} \pm 5\%$ ;  $V_{DD} = 21 \pm .5\text{V}$ )

Symbol	Parameter	Min	Max	Unit	Test Conditions
$V_{DDH}$	$V_{DD}$ Program Voltage High Level	20.5	21.5	V	
$V_{DDL}$	$V_{DD}$ Voltage Low Level	4.75	5.25	V	
$V_{PH}$	$\text{PROG}$ Program Voltage High Level	17.5	18.5	V	
$V_{PL}$	$\text{PROG}$ Voltage Low Level	4.0	$V_{CC}$	V	
$V_{EAH}$	$\text{EA}$ Program or Verify Voltage High Level	17.5	18.5	V	
$I_{DD}$	$V_{DD}$ High Voltage Supply Current		20.0	mA	
$I_{\text{PROG}}$	$\text{PROG}$ High Voltage Supply Current		1.0	mA	
$I_{EA}$	$\text{EA}$ High Voltage Supply Current		1.0	mA	

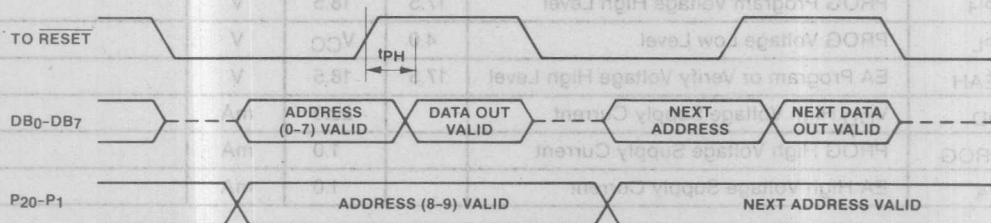


# WAVEFORMS

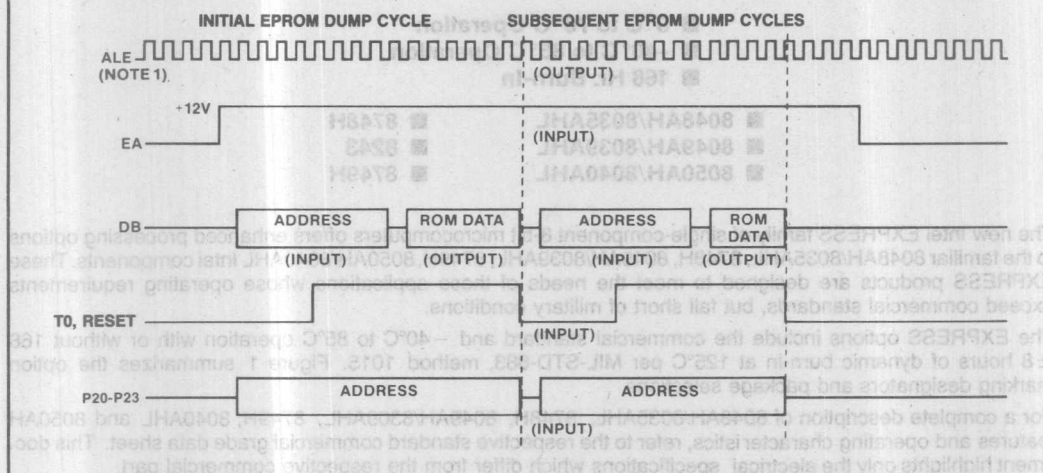
## COMBINATION PROGRAM/VERIFY MODE (EPROM'S ONLY)



## VERIFY MODE



### SUGGESTED EPROM VERIFICATION ALGORITHM FOR HMOS-E DEVICE ONLY



Temp Range °C			Temp Range °C			Temp Range °C		
0 Hz			0 Hz			0 Hz		
-40 to 85			-40 to 85			-40 to 85		
V <sub>SS</sub> = 0V			V <sub>SS</sub> = 0V			V <sub>SS</sub> = 0V		
V <sub>CC</sub> = V <sub>DD</sub> = + 5V			V <sub>CC</sub> = V <sub>DD</sub> = + 5V			V <sub>CC</sub> = V <sub>DD</sub> = + 5V		
ADDR			ADDR			ADDR		
0			0			0		
A10			A10			A10		
0			0			0		
48H			48H			48H		
29H			29H			29H		

**NOTE:** ALE is function of X1, X2 inputs.



H0308H0308H0308H0308

PRELIMINARY

## SINGLE-COMPONENT 8-BIT MICROCOMPUTERS

### EXPRESS

- 0°C to 70°C Operation
- -40°C to 85°C Operation
- 168 Hr. Burn-In

- |                  |         |
|------------------|---------|
| ■ 8048AH/8035AHL | ■ 8748H |
| ■ 8049AH/8039AHL | ■ 8243  |
| ■ 8050AH/8040AHL | ■ 8749H |

The new Intel EXPRESS family of single-component 8-bit microcomputers offers enhanced processing options to the familiar 8048AH/8035AHL, 8748H, 8049AH/8039AHL, 8749H, 8050AH/8040AHL Intel components. These EXPRESS products are designed to meet the needs of those applications whose operating requirements exceed commercial standards, but fall short of military conditions.

The EXPRESS options include the commercial standard and -40°C to 85°C operation with or without 168  $\pm$  8 hours of dynamic burn-in at 125°C per MIL-STD-883, method 1015. Figure 1 summarizes the option marking designators and package selections.

For a complete description of 8048AH/8035AHL, 8748H, 8049AH/8039AHL, 8749H, 8040AHL and 8050AH features and operating characteristics, refer to the respective standard commercial grade data sheet. This document highlights only the electrical specifications which differ from the respective commercial part.

Temp Range C°	0-70	-40-85	0-70	-40-85
	0 Hrs	0 Hrs	168 Hrs	168 Hrs
Burn In				
	P8048AH	TP8048AH	QP8048AH	LP8048AH
	D8048AH	TD8048AH	QD8048AH	LD8048AH
	D8748H	TD8748H	QD8748H	LD8748H
	P8035AHL	TP8035AHL	QP8035AHL	LP8035AHL
	D8035AHL	TD8035AHL	QD8035AHL	LD8035AHL
	P8049AH	TP8049AH	QP8049AH	LP8049AH
	D8049AH	TD8049AH	QD8049AH	LD8049AH
	D8749H	TD8749H	QD8749H	LD8749H
	P8039AHL	TP8039AHL	QP8039AHL	LP8039AHL
	D8039AHL	TD8039AHL	QD8039AHL	LD8039AHL
	P8050AH	TP8050AH	QP8050AH	LP8050AH
	D8050AH	TD8050AH	QD8050AH	LD8050AH
	P8040AHL	TP8040AHL	QP8040AHL	LP8040AHL
	D8040AHL	TD8040AHL	QD8040AHL	LD8040AHL
	P8243	TP8243	QP8243	—
	D8243	TD8243	QD8243	LD8243

\* Commercial Grade  
P Plastic Package  
D Cerdip Package

## Extended Temperature Electrical Specification Deviations\*

TP8048AH/TP8035AHL/LP8048AH/LP8035AHL  
TD8048AH/TD8035AHL/LD8048AH/LD8035AHL

**D.C. CHARACTERISTICS:** ( $T_A = -40^\circ\text{C}$  to  $85^\circ\text{C}$ ;  $V_{CC} = V_{DD} = 5\text{V} \pm 10\%$ ;  $V_{SS} = 0\text{V}$ )

Symbol	Parameter	Limits			Unit	Test Conditions
		Min	Typ	Max		
$V_{IH}$	Input High Voltage (All Except XTAL1, XTAL2, RESET)	2.2		$V_{CC}$	V	
$I_{DD}$	$V_{DD}$ Supply Current		4	8	mA	
$I_{DD} + I_{CC}$	Total Supply Current		40	80	mA	

TP8049AH/TP8039AHL/LP8049AH/LP8039AHL  
TD8049AH/TD8039AHL/LD8049AH/LD8039AHL

**D.C. CHARACTERISTICS:** ( $T_A = -40^\circ\text{C}$  to  $85^\circ\text{C}$ ;  $V_{CC} = V_{DD} = 5\text{V} \pm 10\%$ ;  $V_{SS} = 0\text{V}$ )

Symbol	Parameter	Limits			Unit	Test Conditions
		Min	Typ	Max		
$V_{IH}$	Input High Voltage (All Except XTAL1, XTAL2, RESET)	2.2		$V_{CC}$	V	
$I_{DD}$	$V_{DD}$ Supply Current		5	10	mA	
$I_{DD} + I_{CC}$	Total Supply Current		50	100	mA	

TP8050AH/TP8040AHL/LP8050AH/LP8040AHL  
TD8050AH/TD8040AHL/LD8050AH/LD8040AHL

**D.C. CHARACTERISTICS:** ( $T_A = -40^\circ\text{C}$  to  $85^\circ\text{C}$ ;  $V_{CC} = V_{DD} = 5\text{V} \pm 10\%$ ;  $V_{SS} = 0\text{V}$ )

Symbol	Parameter	Limits			Unit	Test Conditions
		Min	Typ	Max		
$V_{IH}$	Input High Voltage (All Except XTAL1, XTAL2, RESET)	2.2		$V_{CC}$	V	
$I_{DD}$	$V_{DD}$ Supply Current		10	20	mA	
$I_{DD} + I_{CC}$	Total Supply Current		75	120	mA	



Extended Temperature Electrical Specification Deviations\*

TD8748H/LD8748H

**D.C. CHARACTERISTICS:** ( $T_A = -40^\circ\text{C}$  to  $85^\circ\text{C}$ ;  $V_{CC} = V_{DD} = 5V \pm 10\%$ ;  $V_{SS} = 0V$ )

Symbol	Parameter	Limits			Unit	Test Conditions
		Min	Typ	Max		
$V_{IH}$	Input High Voltage (All Except XTAL1, XTAL2, RESET)	2.2		$V_{CC}$	V	
$I_{DD} + I_{CC}$	Total Supply Current		50	130	mA	

TD8749H/LD8749H

**D.C. CHARACTERISTICS:** ( $T_A = -40^\circ\text{C}$  to  $85^\circ\text{C}$ ;  $V_{CC} = V_{DD} = 5V \pm 10\%$ ;  $V_{SS} = 0V$ )

Symbol	Parameter	Limits			Unit	Test Conditions
		Min	Typ	Max		
$V_{IH}$	Input High Voltage (All Except XTAL1, XTAL2, RESET)	2.2		$V_{CC}$	V	
$I_{DD} + I_{CC}$	Total Supply Current		75	150	mA	

TP8243/TD8243/LD8243

**D.C. CHARACTERISTICS:** ( $T_A = -40^\circ\text{C}$  to  $85^\circ\text{C}$ ;  $V_{CC} = 5V \pm 10\%$ ;  $V_{SS} = 0V$ )

Symbol	Parameter	Limits			Unit	Test Conditions
		Min	Typ	Max		
$I_{CC}$	$V_{CC}$ Supply Current		15	25	mA	
$V_{IH}$	Input High Voltage (All Except XTAL1, XTAL2, RESET)	2.2		$V_{CC}$	V	
$I_{DD}$	$V_{DD}$ Supply Current		10	20	mA	
$I_{DD} + I_{CC}$	Total Supply Current		13	150	mA	

## Extended Temperature Electrical Specification Deviations\*

TD8022/LD8022

**D.C. CHARACTERISTICS:** ( $T_A = -40^\circ\text{C}$  to  $85^\circ\text{C}$ ;  $V_{CC} = 5.5\text{V} \pm 1\text{V}$ ;  $V_{SS} = 0\text{V}$ )

Symbol	Parameter	Limits			Unit	Test Conditions
		Min	Typ	Max		
$V_{IL1}$	Input Low Voltage (Port 0)	-0.5		$V_{TH}-0.2$	V	
$V_{IH}$	Input High Voltage (All Except XTAL1, RESET)	2.3		$V_{CC}$	V	$V_{CC} = 5.0\text{V} \pm 10\%$ $V_{TH}$ Floating
$V_{IH1}$	Input High Voltage (All Except XTAL1, RESET)	3.8		$V_{CC}$	V	$V_{CC} = 5.5\text{V} \pm 1\text{V}$ $V_{TH}$ Floating
$V_{IH2}$	Input High Voltage (Port 0)	$V_{TH}+0.2$		$V_{CC}$	V	
$V_{IH3}$	Input High Voltage (RESET, XTAL1)	3.8		$V_{CC}$	V	
$V_{IL}$	Input Low Voltage	-0.5		0.5	V	
$V_{OL}$	Output Low Voltage			0.45	V	$I_{OL} = 0.8\text{ mA}$
$V_{OL1}$	Output Low Voltage (P10, P11)			2.5	V	$I_{OL} = 3\text{ mA}$
$V_{OH}$	Output High Voltage (All unless open drain option Port 0)	2.4			V	$I_{OH} = 30\text{ }\mu\text{A}$
$I_{LI}$	Input Current (T1)			$\pm 700$	$\mu\text{A}$	$V_{CC} \geq V_{IN} \geq$ $V_{SS} + 0.45\text{V}$
$I_{LI1}$	Input Current to Ports			500	$\mu\text{A}$	$V_{IN} = 0.45\text{V}$
$I_{CC}$	$V_{CC}$ Supply Current			120	mA	

**A.C. CHARACTERISTICS:** ( $T_A = -40^\circ\text{C}$  to  $85^\circ\text{C}$ ;  $V_{CC} = 5.5\text{V} \pm 1\text{V}$ ;  $V_{SS} = 0\text{V}$ )

Symbol	Parameter	Min	Max	Unit	Test Conditions
$t_{CY}$	Cycle Time	8.38	50.0	$\mu\text{s}$	3.58 MHz XTAL = $8.38\text{ }\mu\text{s } t_{CY}$
$V_{T1}$	Zero-Cross Detection Input (T1)	1	3	VACpp	AC Coupled
AZC	Zero-Cross Accuracy		$\pm 200$	mV	60 Hz Sine Wave
$F_{T1}$	Zero-Cross Detection Input Frequency (T1)	0.05	1	kHz	
$t_{LL}$	ALE Pulse Width	3.9	23.0	$\mu\text{s}$	$t_{CY} = 8.38\text{ }\mu\text{s}$ for min

NOTE: Control Outputs:  $C_L = 80\text{ pf}$ ;  $T_{CY} = 8.38\text{ }\mu\text{sec}$ .**A/D CONVERTER CHARACTERISTICS:** ( $AV_{CC} = 5.5\text{V} \pm 1\text{V}$ ;  $AV_{SS} = 0\text{V}$ ;  $AV_{CC}/2 \leq V_{AREF} \leq AV_{CC}$ )

Parameter	Limits			Unit	Test Conditions
	Min	Typ	Max		
Absolute Accuracy			$1.6\% \text{ FSR} \pm \frac{1}{2} \text{ LSB}$	LSB	

NOTE: The analog input must be maintained at a constant voltage during the sample time ( $t_{ss} + t_{sh}$ ).

\*Refer to individual commercial grade data sheets for complete operating characteristics.

# 80C49-7/80C39-7

## CHMOS SINGLE-COMPONENT 8-BIT MICROCOMPUTER

- 80C49-7 Low Power Mask Programmable ROM
- 80C39-7 Low Power, CPU only

- Pin-to-pin Compatible with Intel's 8049AH/8039AHL
- 1.36  $\mu$ sec Instruction Cycle. All Instructions 1 or 2 Cycles
- Ability to Maintain Operation during AC Power Line Interruptions
- Exit Idle Mode with an External or Internal Interrupt Signal
- Battery Operation
- 3 Power Consumption Selections
  - Normal Operation: 12 mA @ 11 MHz @ 5V
  - Idle Mode: 5 mA @ 11 MHz @ 5V
  - Power Down: 2  $\mu$ A @ 2.0V
- 11 MHz, TTL Compatible Operation:  $V_{CC} = 5V \pm 10\%$
- CMOS Compatible Operation:  $V_{CC} = 5V \pm 20\%$

Intel's 80C49-7/80C39-7 are low power, CHMOS versions of the popular MCS<sup>®</sup>-48 HMOS family members. CHMOS is a technology built on HMOS II and features high resistivity P substrate, diffused N well, and scaled N and P channel devices. The 80C49-7/80C39-7 have been designed to provide low power consumption and high performance.

The 80C49-7 contains a 2K x 8 program memory, a 128 x 8 x 8 RAM data memory, 27 I/O lines, and an 8-bit timer/counter in addition to an on-board oscillator and clock circuits. For systems that require extra capability, the 80C49-7 can be expanded using CMOS external memories and MCS<sup>®</sup>-80 and MCS<sup>®</sup>-85 peripherals. The 80C39-7 is the equivalent of the 80C49-7 without program memory on-board.

The CHMOS design of the 80C49-7 opens new application areas that require battery operation, low power standby, wide voltage range, and the ability to maintain operation during AC power line interruptions. These applications include portable and hand-held instruments, telecommunications, consumer, and automotive.

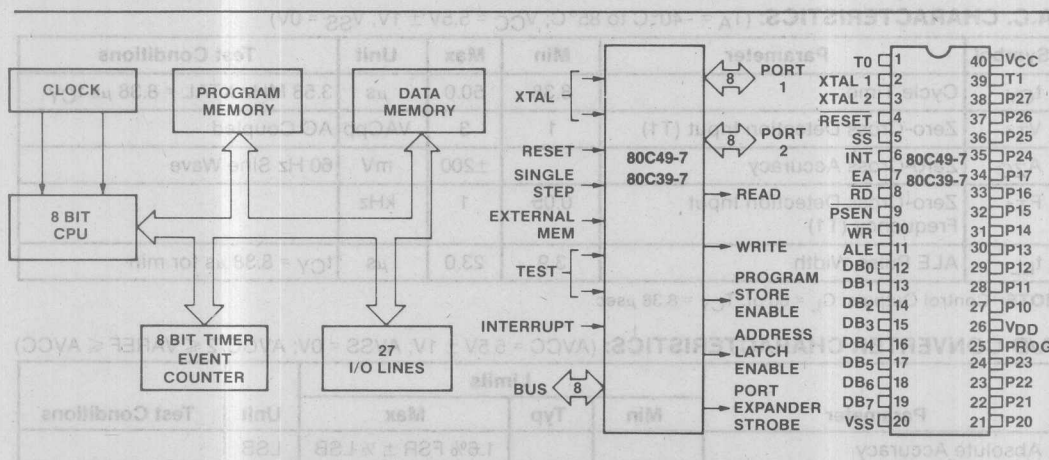


Figure 1.  
Block Diagram

Figure 2.  
Logic Symbol

Figure 3.  
Pin Configuration

Table 1. Pin Description

Symbol	Pin No.	Function
V <sub>SS</sub>	20	Circuit GND potential
V <sub>DD</sub>	26	Low Power standby pin
V <sub>CC</sub>	40	Main power supply; +5V during operation.
PROG	25	Output strobe for 82C43 I/O expander.
P10-P17 Port 1	27-34	8-bit quasi-bidirectional port.
P20-P23	21-24	8-bit quasi-bidirectional port.
P24-P27 Port 2	35-38	P20-P23 contain the four high order program counter bits during an external program memory fetch and serve as a 4-bit I/O expander bus for 8243.
DB0-DB7 BUS	12-19	True bidirectional port which can be written or read synchronously using the RD, WR strobes. The port can also be statically latched.  Contains the 8 low order program counter bits during an external program memory fetch, and receives the addressed instruction under the control of PSEN. Also contains the address and data during an external RAM data store instruction, under control of ALE, RD, and WR.
T0	1	Input pin testable using the conditional transfer instructions JT0 and JNT0. T0 can be designated as a clock output using ENT0 CLK instruction.
T1	39	Input pin testable using the JT1, and JNT1 instructions.

Symbol	Pin No.	Function
		Can be designated the timer/counter input using the STRT CNT instruction.
INT	6	Interrupt input. Initiates an interrupt if interrupt is enabled. Interrupt is disabled after a reset. Also testable with conditional jump instruction. (Active low) Interrupt must remain low for at least 3 machine cycles for proper operation.
RD	8	Output strobe activated during a BUS read. Can be used to enable data onto the bus from an external device.  Used as a read strobe to external data memory. (Active low)
RESET	4	Input which is used to initialize the processor. (Active low) (Non TTL V <sub>IH</sub> )
WR	10	Output strobe during a bus write. (Active low)  Used as write strobe to external data memory.
ALE	11	Address latch enable. This signal occurs once during each cycle and is useful as a clock output.  The negative edge of ALE strobes address into external data and program memory.
PSEN	9	Program store enable. This output occurs only during a fetch to external program memory. (Active low)
SS	5	Single step input can be used in conjunction with



Table 1. Pin Description (Continued)

Symbol	Pin No.	Function
SS (Con't)	6	ALE to "single step" the processor through each instruction (Active low)
EA	7	External access input which forces all program memory fetches to reference external memory. Useful for emulation and debug, and essential for testing

Symbol	Pin No.	Function
		and program verification. (Active high)
XTAL1	2	One side of crystal input for internal oscillator. Also input for external source. (Non TTL $V_{IH}$ )
XTAL2	3	Other side of crystal input.

### IDLE MODE DESCRIPTION

The 80C49-7, when placed into Idle mode, keeps the oscillator, the internal timer and the external interrupt and counter pins functioning and maintains the internal register and RAM status.

To place the 80C49-7 in Idle mode, a command instruction (op code 01H) is executed. To terminate Idle mode, a reset must be performed or interrupts must be enabled and an interrupt signal generated. There are two interrupt sources that can restore normal operation. One is an external signal applied to the interrupt pin. The other is from the overflow of the timer/counter. When either interrupt is invoked, the CPU is taken out of Idle mode and vectors to the interrupt's service routine address. Along with the Idle mode, the standard MCS®-48 power-down mode is still maintained.

RESET	4	input which is used to initialize the processor (Active low) (Non TTL $V_{IH}$ )
WR	10	Output strobe during a bus write (Active low) Used as write strobe to external data memory
ALE	11	Address latch enable. This signal occurs once during each cycle and is useful as a clock output. The negative edge of ALE stores address info external data and program memory
PSEN	9	Program store enable. This output occurs only during a fetch to external program memory (Active low)
SS	6	Single step input can be used in conjunction with

		read synchronously using the RD, WR strobes. The port can also be statically latched.
		Contains the 8 low order program counter bits during an external program memory fetch, and receives the addressed instruction under the control of PSEN. Also contains the address and data during an external RAM data store instruction under control of ALE, RD, and WR
IO/M <sub>1</sub>	1	Input pin testable using the conditional transfer instructions JTO and JNT. IO can be designated as a clock output using ENTO CLK instruction.
INT1	20	Input pin testable using the INT1 and INT1 instructions

Table 2. Instruction Set

Accumulator			
Mnemonic	Description	Bytes	Cycles
ADD A, R	Add register to A	1	1
ADD A, @R	Add data memory to A	1	1
ADD A, # data	Add immediate to A	2	2
ADDC A, R	Add register with carry	1	1
ADDC A, @R	Add data memory with carry	1	1
ADDC A, # data	Add immediate with carry	2	2
ANL A, R	And register to A	1	1
ANL A, @R	And data memory to A	1	1
ANL A, # data	And immediate to A	2	2
ORL A, R	Or register to A	1	1
ORL A, @R	Or data memory to A	1	1
ORL A, # data	Or immediate to A	2	2
XRL A, R	Exclusive or register to A	1	1
XRL A, @R	Exclusive or data memory to A	1	1
XRL A, # data	Exclusive or immediate to A	2	2
INC A	Increment A	1	1
DEC A	Decrement A	1	1
CLR A	Clear A	1	1
CPL A	Complement A	1	1
DA A	Decimal adjust A	1	1
SWAP A	Swap nibbles of A	1	1
RL A	Rotate A left	1	1
RLC A	Rotate A left through carry	1	1
RR A	Rotate A right	1	1
RRC A	Rotate A right through carry	1	1

Input/Output			
Mnemonic	Description	Bytes	Cycles
IN A, P	Input port to A	1	2
OUTL P, A	Output A to port	1	2
ANL P, # data	And immediate to port	2	2
ORL P, # data	Or immediate to port	2	2
INS A, BUS	Input BUS to A	1	2
OUTL BUS, A	Output A to BUS	1	2
ANL BUS, # data	And immediate to BUS	2	2
ORL BUS, # data	Or immediate to BUS	2	2
MOVD A, P	Input expander port to A	1	2
MOVD P, A	Output A to expander port	1	2
ANLD P, A	And A to expander port	1	2
ORLD P, A	Or A to expander port	1	2

Registers			
Mnemonic	Description	Bytes	Cycles
INC R	Increment register	1	1
INC @R	Increment data memory	1	1
DEC R	Decrement register	1	1

Branch			
Mnemonic	Description	Bytes	Cycles
JMP addr	Jump unconditional	2	2
JMPP @A	Jump indirect	1	2
DJNZ R, addr	Decrement register and skip	2	2
JC addr	Jump on carry = 1	2	2
JNC addr	Jump on carry = 0	2	2
JZ addr	Jump on A zero	2	2
JNZ addr	Jump on A not zero	2	2
JT0 addr	Jump on T0 = 1	2	2
JNT0 addr	Jump on T0 = 0	2	2
JT1 addr	Jump on T1 = 1	2	2
JNT1 addr	Jump on T1 = 0	2	2
JF0 addr	Jump on F0 = 1	2	2
JF1 addr	Jump on F1 = 1	2	2
JTF addr	Jump on timer flag	2	2
JNI addr	Jump on INT = 0	2	2
JBb addr	Jump on accumulator bit	2	2

Subroutine			
Mnemonic	Description	Bytes	Cycles
CALL addr	Jump to subroutine	2	2
RET	Return	1	2
RETR	Return and restore status	1	2

Flags			
Mnemonic	Description	Bytes	Cycles
CLR C	Clear carry	1	1
CPL C	Complement carry	1	1
CLR F0	Clear flag 0	1	1
CPL F0	Complement flag 0	1	1
CLR F1	Clear flag 1	1	1
CPL F1	Complement flag 1	1	1

Table 2. Instruction Set (Continued)

Data Moves			
Mnemonic	Description	Bytes	Cycles
MOV A, R	Move register to A	1	1
MOV A, @R	Move data memory to A	1	1
MOV A, # data	Move immediate to A	2	2
MOV R, A	Move A to register	1	1
MOV @R, A	Move A to data memory	1	1
MOV R, # data	Move immediate to register	2	2
MOV @R, # data	Move immediate to data memory	2	2
MOV A, PSW	Move PSW to A	1	1
MOV PSW, A	Move A to PSW	1	1
XCH A, R	Exchange A and register	1	1
XCH A, @R	Exchange A and data memory	1	1
XCHD A, @R	Exchange nibble of A and register	1	1
MOVX A, @R	Move external data memory to A	1	2
MOVX @R, A	Move A to external data memory	1	2
MOVP A, @A	Move to A from current page	1	2
MOVP3 A, @A	Move to A from page 3	1	2

Timer/Counter			
Mnemonic	Description	Bytes	Cycles
MOV A, T	Read timer/counter	1	1
MOV T, A	Load timer/counter	1	1
STRT T	Start timer	1	1
STRT CNT	Start counter	1	1
STOP TCNT	Stop timer/counter	1	1
EN TCNTI	Enable timer/counter interrupt	1	1
DIS TCNTI	Disable timer/counter interrupt	1	1

Control			
Mnemonic	Description	Bytes	Cycles
EN I	Enable external interrupt	1	1
DIS I	Disable external interrupt	1	1
SEL RB0	Select register bank 0	1	1
SEL RB1	Select register bank 1	1	1
SEL MB0	Select memory bank 0	1	1
SEL MB1	Select memory bank 1	1	1
ENTO CLK	Enable clock output on T0	1	1

Mnemonic	Description	Bytes	Cycles
NOP	No operation	1	1
IDL	Select Idle Operation	1	1

**ABSOLUTE MAXIMUM RATINGS\***

Ambient Temperature Under Bias . . . . . 0°C to 70°C  
 Storage Temperature . . . . . -65°C to +150°C  
 Voltage On Any Pin With Respect to Ground . . . . . -0.5V to  $V_{CC} + 1V$   
 Maximum Voltage On Any Pin With Respect to Ground . . . . . 7V  
 Power Dissipation . . . . . 1.0 Watt

\*NOTICE: Stresses above those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of device at these or any other conditions above those indicated in the operational sections of this specification is not implied.

**D.C. CHARACTERISTICS:** ( $T_A = 0^\circ\text{C}$  to  $70^\circ\text{C}$ ;  $V_{CC} = V_{DD} = 5\text{V} \pm 20\%$ ;  $|V_{CC} - V_{DD}| \leq 1.5\text{V}$ ;  
 $V_{SS} = 0\text{V}$ )

Symbol	Parameter	Limits			Unit	Test Conditions
		Min	Typ	Max		
$V_{IL}$	Input Low Voltage (All Except X1, RESET)	$-0.5$		$.18 V_{CC}$	V	
$V_{IL1}$	Input Low Voltage X1, RESET	$-5$		$.13 V_{CC}$	V	
$V_{IH}$	Input High Voltage (All Except XTAL1, RESET)	$0.2 V_{CC} + 1.2$		$V_{CC}$	V	
$V_{IH1}$	Input High Voltage (X1, RESET)	$.7 V_{CC}$		$V_{CC}$	V	
$V_{OL}$	Output Low Voltage (BUS)			$.6$	V	$I_{OL} = 2.0\text{ mA}$
$V_{OL1}$	Output Low Voltage (RD, WR, PSEN, ALE)			$.6$	V	$I_{OL} = 1.8\text{ mA}$
$V_{OL2}$	Output Low Voltage (PROG)			$.6$	V	$I_{OL} = 1.0\text{ mA}$
$V_{OL3}$	Output Low Voltage (All Other Outputs)			$.6$	V	$I_{OL} = 1.6\text{ mA}$
$V_{OH}$	Output High Voltage (BUS)	$.75 V_{CC}$			V	$I_{OH} = -400\text{ }\mu\text{A}$
$V_{OH1}$	Output High Voltage (RD, WR, PSEN, ALE)	$.75 V_{CC}$			V	$I_{OH} = -100\text{ }\mu\text{A}$
$V_{OH2}$	Output High Voltage (All Other Outputs)	$2.4$ $3.0$			V	$I_{OH} = -40\text{ }\mu\text{A}$ $I_{OH} = -20\text{ }\mu\text{A}$
$I_{L1}$	Input Leakage Current (T1, INT, EA)			$\pm 5$	$\mu\text{A}$	$V_{SS} \leq V_{IN} \leq V_{CC}$
$I_{L11}$	Input Leakage Current (P10-P17, P20-P27, SS)			$-500$	$\mu\text{A}$	$V_{SS} \leq V_{IN} \leq V_{CC}$
$I_{LO}$	Output Leakage Current (BUS, TO) (High Impedance State)			$\pm 5$	$\mu\text{A}$	$V_{SS} \leq V_{IN} \leq V_{CC}$
$I_{LR}$	Input Leakage Current (RESET)	$-10$		$-300$	$\mu\text{A}$	$V_{SS} \leq V_{IN} \leq V_{IH1}$
$I_{PD}$	Power Down Standby Current			$2$	$\mu\text{A}$	$V_{DD} = 2.0\text{V}$ RESET $\leq V_{IL}$

 **$I_{CC}$  Active Current (mA)**

$V_{CC}$	4V	5V	6V
1 MHz	2.5	3.3	4.0
6 MHz	5	6.8	8.5
11 MHz	9	12	15

 **$I_{CC}$  Idle Current (mA)**

$V_{CC}$	4V	5V	6V
1 MHz	1.7	2.0	2.2
6 MHz	2	3	4
11 MHz	3.5	4.8	6

**Absolute Maximum Unloaded Current** **$I_{CC}$  Test Conditions:** **$I_{CC}$  Active**

All outputs disconnected  
 T1, INT, SS, T0 connected to HIGH ( $V_{IH}$ )  
 EA, RST connected to LOW ( $V_{IL}$ )  
 XTAL1 External Drive  
 Rise Time = 10 ns, Fall Time = 10 ns  
 XTAL2 No connection  
 $V_{IH} = V_{CC} - 0.5\text{V}$   
 $V_{IL} = V_{SS} + 0.5\text{V}$

 **$I_{CC}$  Idle**

All outputs disconnected  
 XTAL1 External Drive  
 Rise Time = 10 ns, Fall Time = 10 ns  
 XTAL2 No connection  
 $V_{IH} = V_{CC} - 0.5\text{V}$   
 $V_{IL} = V_{SS} + 0.5\text{V}$



**A.C. CHARACTERISTICS:** ( $T_A = 0^\circ\text{C}$  to  $70^\circ\text{C}$ ;  $V_{CC} = V_{DD} = 5V \pm 20\%$ ;  $|V_{CC} - V_{DD}| \leq 1.5V$ ;  $V_{SS} = 0V$ )

Symbol	Parameter	f (t) (Note 3)	11 MHz		Unit	Conditions (Note 1)
			Min	Max		
t	Clock Period	1/xtal freq	90.9	1000	ns	(Note 3)
t <sub>LL</sub>	ALE Pulse Width	3.5t-170	150		ns	
t <sub>AL</sub>	Addr Setup to ALE	2t-110	70		ns	(Note 2)
t <sub>LA</sub>	Addr Hold from ALE	t-40	50		ns	
t <sub>CC1</sub>	Control Pulse Width ( $\overline{RD}$ , $\overline{WR}$ )	7.5t-200	480		ns	
t <sub>CC2</sub>	Control Pulse Width ( $\overline{PSEN}$ )	6t-200	350		ns	
t <sub>DW</sub>	Data Setup before $\overline{WR}$	6.5t-200	390		ns	
t <sub>WD</sub>	Data Hold after $\overline{WR}$	t-50	40		ns	
t <sub>DR</sub>	Data Hold ( $\overline{RD}$ , $\overline{PSEN}$ )	1.5t-30	0	110	ns	
t <sub>RD1</sub>	$\overline{RD}$ to Data in	6t-170		350	ns	
t <sub>RD2</sub>	$\overline{PSEN}$ to Data in	4.5t-170		190	ns	
t <sub>AW</sub>	Addr Setup to $\overline{WR}$	5t-150	300		ns	
t <sub>AD1</sub>	Addr Setup to Data ( $\overline{RD}$ )	10.5t-220		730	ns	
t <sub>AD2</sub>	Addr Setup to Data ( $\overline{PSEN}$ )	7.5t-220		460	ns	
t <sub>AFC1</sub>	Addr Float to $\overline{RD}$ , $\overline{WR}$	2t-40	140		ns	(Note 2)
t <sub>AFC2</sub>	Addr Float to $\overline{PSEN}$	.5t-40	10		ns	(Note 2)
t <sub>LAFC1</sub>	ALE to Control ( $\overline{RD}$ , $\overline{WR}$ )	3t-75	200		ns	
t <sub>LAFC2</sub>	ALE to Control ( $\overline{PSEN}$ )	1.5t-75	60		ns	
t <sub>CA1</sub>	Control to ALE ( $\overline{RD}$ , $\overline{WR}$ , $\overline{PROG}$ )	t-65	25		ns	
t <sub>CA2</sub>	Control to ALE ( $\overline{PSEN}$ )	4t-70	290		ns	
t <sub>CP</sub>	Port Control Setup to $\overline{PROG}$	1.5t-80	50		ns	
t <sub>PC</sub>	Port Control Hold to $\overline{PROG}$	4t-260	100		ns	
t <sub>PR</sub>	$\overline{PROG}$ to P2 Input Valid	8.5t-120		650	ns	
t <sub>PF</sub>	Input Data Hold from $\overline{PROG}$	1.5t	0	140	ns	
t <sub>DP</sub>	Output Data Setup	6t-290	250		ns	
t <sub>PD</sub>	Output Data Hold	1.5t-90	40		ns	
t <sub>PP</sub>	$\overline{PROG}$ Pulse Width	10.5t-250	700		ns	
t <sub>PL</sub>	Port 2 I/O Setup to ALE	4t-200	160		ns	
t <sub>LP</sub>	Port 2 I/O Hold to ALE	1.5t-120	15		ns	
t <sub>PV</sub>	Port Output from ALE	4.5t+100		510	ns	
t <sub>OPRR</sub>	T0 Rep Rate	3t	270		ns	
t <sub>CY</sub>	Cycle Time	15t	1.36	15.0	$\mu\text{s}$	

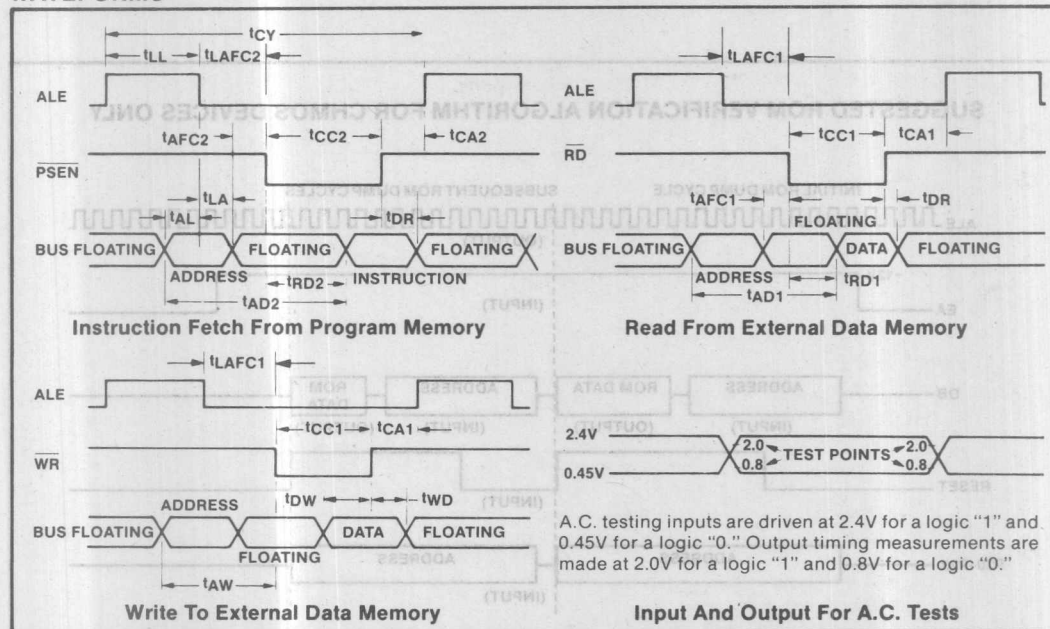
**Notes:**

1. Control Outputs CL = 80pF  
BUS Outputs CL = 150pF

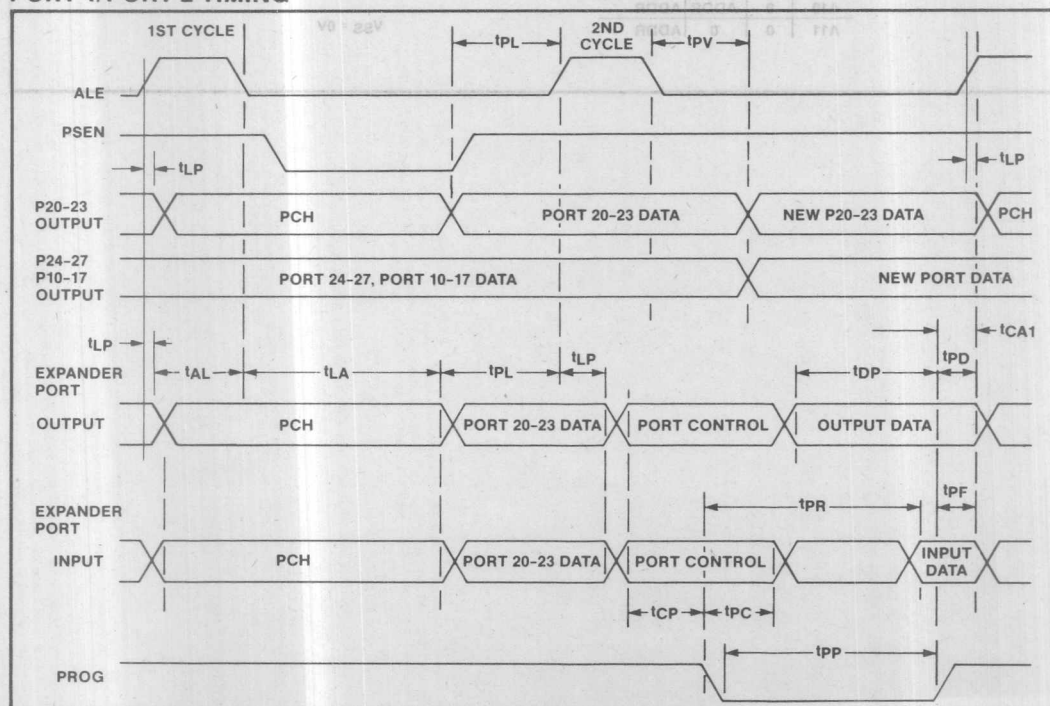
2. BUS High Impedance  
Load 20pF

3. f(t) assumes 50% duty cycle on X1, X2. Max  
clock period is for a 1 MHz crystal input.

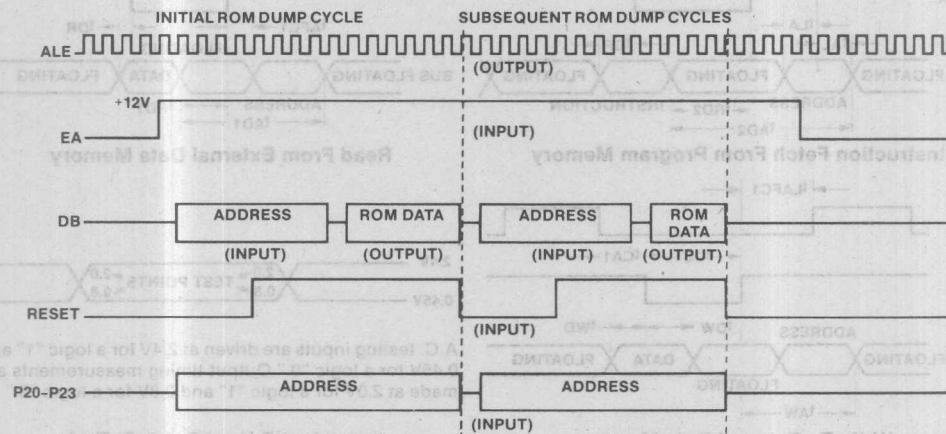
# WAVEFORMS



## PORT 1/PORT 2 TIMING



# SUGGESTED ROM VERIFICATION ALGORITHM FOR CHMOS DEVICES ONLY



	C48	C49	C50
A10	0	ADDR	ADDR
A11	0	0	ADDR

$V_{CC} = V_{DD} = +5V$

$V_{SS} = 0V$

---

# The RUPI™-44 Family

---

# 16



16

The RUP™-44 Family

---

## CHAPTER 16

### THE RUPi™-44 FAMILY: MICROCONTROLLER WITH ON-CHIP COMMUNICATION CONTROLLER

#### 16.0 INTRODUCTION

The RUPi-44 family is designed for applications requiring local intelligence at remote nodes, and communication capability among these distributed nodes. The RUPi-44 integrates onto a single chip Intel's highest performance microcontroller, the 8051-core, with an intelligent and high performance Serial communication controller, called the Serial Interface Unit, or SIU. See Figure 16-1. This dual controller architecture allows complex control and high speed data communication functions to be realized cost effectively.

The RUPi-44 family consists of three pin compatible parts:

- 8344—8051 Microcontroller with SIU
- 8044—An 8344 with 4K bytes of on-chip ROM program memory.
- 8744—An 8344 with 4K bytes of on-chip EPROM program memory.

#### 16.1 ARCHITECTURE OVERVIEW

The 8044's dual controller architecture enables the RUPi to perform complex control tasks and high speed communication in a distributed network environment.

The 8044 microcontroller is the 8051-core, and maintains complete software compatibility with it. The microcontroller contains a powerful CPU with on-chip peripherals, making it capable of serving sophisticated

real-time control applications such as instrumentation, industrial control, and intelligent computer peripherals. The microcontroller features on-chip peripherals such as two 16-bit timer/counters and 5 source interrupt capability with programmable priority levels. The microcontroller's high performance CPU executes most instructions in 1 microsecond, and can perform an  $8 \times 8$  multiply in 4 microseconds. The CPU features a Boolean processor that can perform operations on 256 directly addressable bits. 192 bytes of on-chip data RAM can be extended to 64K bytes externally. 4K bytes of on-chip program ROM can be extended to 64K bytes externally. The CPU and SIU run concurrently. See Figure 16-2.

The SIU is designed to perform serial communications with little or no CPU involvement. The SIU supports data rates up to 2.4Mbps, externally clocked, and 375K bps self clocked (i.e., the data clock is recovered by an on-chip digital phase locked loop). SIU hardware supports the HDLC/SDLC protocol: zero bit insertion/deletion, address recognition, cyclic redundancy check, and frame number sequence check are automatically performed.

The SIU's Auto mode greatly reduces communication software overhead. The AUTO mode supports the SDLC Normal Response Mode, by performing secondary station responses in hardware without any CPU involvement. The Auto mode's interrupt control and frame sequence numbering capability eliminates software overhead normally required in conventional systems. By using the Auto mode, the CPU is free to concentrate on real time control of the application.

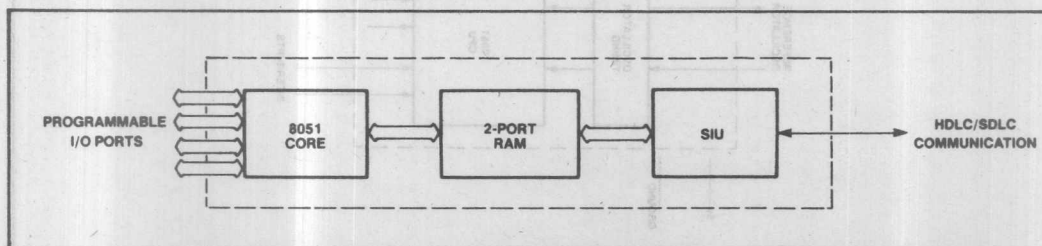


Figure 16-1. RUPi™-44 Dual Controller Architecture

# УЛНААХ АА-71УН ЭНТ АН-НО НТНВ РЕЛЛЭТНОСОРДИМ РЕЛЛЭТНОС ИОИТ АДИИУННОС

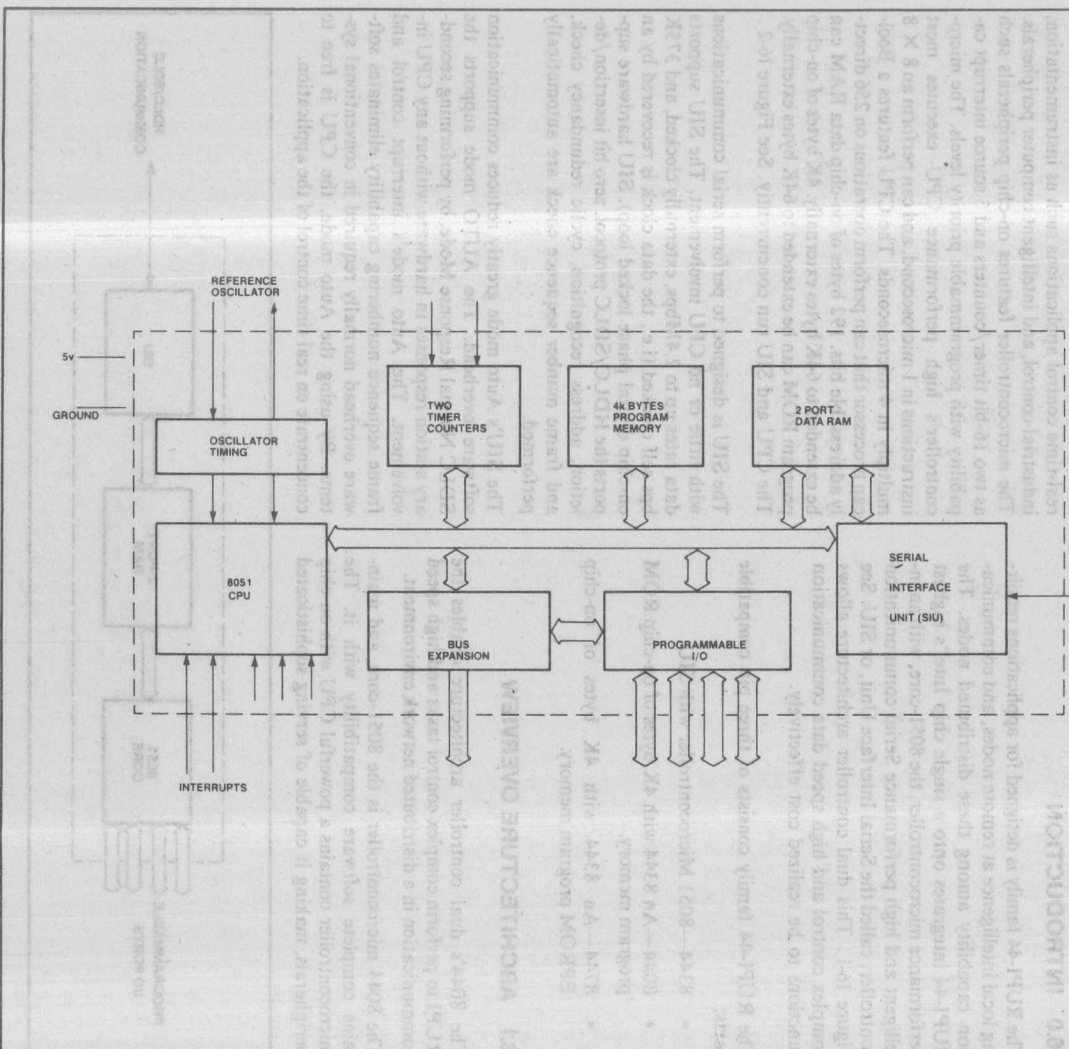


Figure 16-2. Simplified 8044 Block Diagram

## 16.2 THE HDLC/SDLC PROTOCOLS

### 16.2.1 HDLC/SDLC Advantages over Async

The High Level Data Link Control, HDLC, is a standard communication link control established by the International Standards Organization (ISO). SDLC is a subset of HDLC.

HDLC and SDLC are both well recognized standard serial protocols. The Synchronous Data Link Control, SDLC, is an IBM standard communication protocol. IBM originally developed SDLC to provide efficient, reliable and simple communication between terminals and computers.

The major advantages of SDLC/HDLC over Asynchronous communications protocol (Async):

- **SIMPLE:** Data Transparency

- **EFFICIENT:** Well Defined Message-Level Operation

- **RELIABLE:** Frame Check Sequence and Frame Numbering

The SDLC reduces system complexity. HDLC/SDLC are "data transparent" protocols. Data transparency means that an arbitrary data stream can be sent without concern that some of data could be mistaken for a protocol controller. Data transparency relieves the communication controller having to detect special characters.

SDLC/HDLC provides more data throughput than Async. SDLC/HDLC runs at Message-level Operation which transmits multiple bytes within the frame. Whereas Async is based on character-level operation. Async transmits or receives a character at a time. Since Async requires start and stop bits in every transmission, there is a considerable waste of overhead compared to SDLC/HDLC.

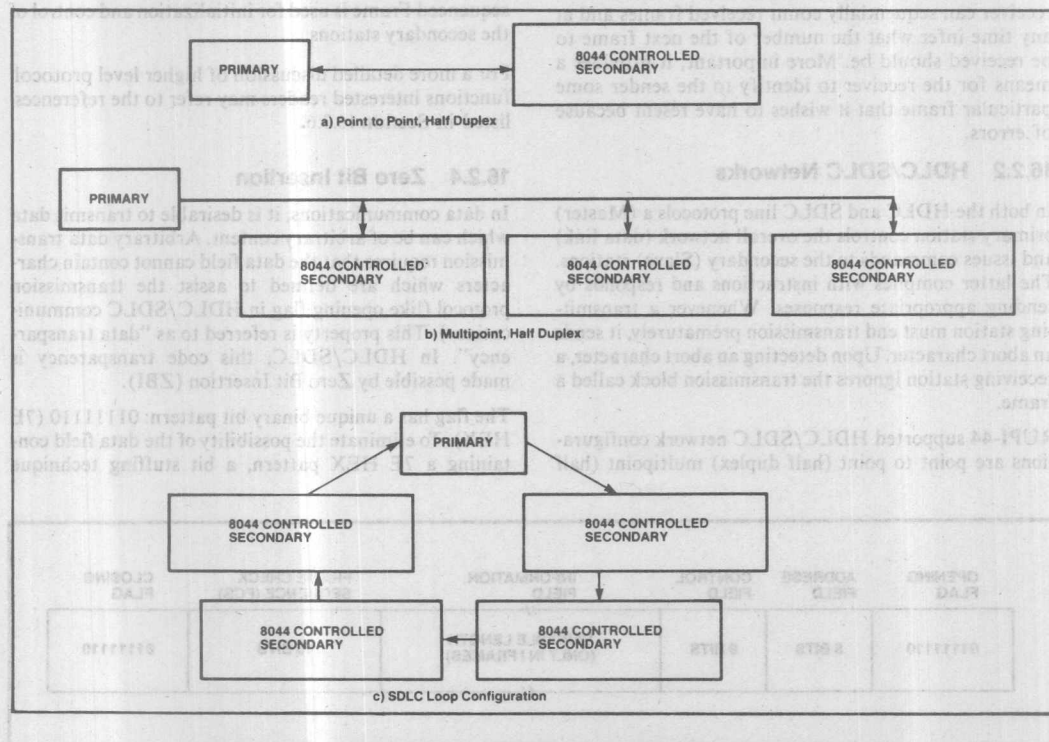


Figure 16-3. RUP1™-44 Supported Network Configurations



Due to SDLC/HDLC's well delineated field (see Figure 16-4) the CPU does not have to interpret character by character to determine control field and information field. In the case of Async, CPU must look at each character to interpret what it means. The practical advantage of such feature is straight forward use of DMA for information transfer.

In addition, SDLC/HDLC further improves Data throughput using implied Acknowledgement of transferred information. A station using SDLC/HDLC may acknowledge previously received information while transmitting different information in the same frame. In addition, up to 7 messages may be outstanding before an acknowledgement is required.

The HDLC/SDLC protocol can be used to realize reliable data links. Reliable Data transmission is ensured at the bit level by sending a frame check sequence, cyclic redundancy checking, within the frame. Reliable frame transmission is ensured by sending a frame number identification with each frame. This means that a receiver can sequentially count received frames and at any time infer what the number of the next frame to be received should be. More important, it provides a means for the receiver to identify to the sender some particular frame that it wishes to have resent because of errors.

### 16.2.2 HDLC/SDLC Networks

In both the HDLC and SDLC line protocols a (Master) primary station controls the overall network (data link) and issues commands to the secondary (Slave) stations. The latter complies with instructions and responds by sending appropriate responses. Whenever a transmitting station must end transmission prematurely, it sends an abort character. Upon detecting an abort character, a receiving station ignores the transmission block called a frame.

RUP1-44 supported HDLC/SDLC network configurations are point to point (half duplex) multipoint (half

duplex), and loop. In the loop configuration the stations themselves act as repeaters, so that long links can be easily realized, see Figure 16-3.

### 16.2.3 Frames

An HDLC/SDLC frame consists of five basic fields: Flag, Address, Control, Data and Error Detection. A frame is bounded by flags—opening and closing flags. An address field is 8 bits wide in SDLC, extendable to 2 or more bytes in HDLC. The control field is also 8 bits wide, extendable to two bytes in HDLC. The SDLC data field or information field may be any number of bytes. The HDLC data field may or may not be on an 8 bit boundary. A powerful error detection code called Frame Check Sequence contains the calculated CRC (Cycle Redundancy Code) for all the bits between the flags. See Figure 16-4.

In HDLC and SDLC are three types of frames; an Information Frame is used to transfer data, a Supervisory Frame is used for control purposes, and a Non-sequenced Frame is used for initialization and control of the secondary stations.

For a more detailed discussion of higher level protocol functions interested readers may refer to the references listed in Section 16.2.6.

### 16.2.4 Zero Bit Insertion

In data communications, it is desirable to transmit data which can be of arbitrary content. Arbitrary data transmission requires that the data field cannot contain characters which are defined to assist the transmission protocol (like opening flag in HDLC/SDLC communications). This property is referred to as "data transparency". In HDLC/SDLC, this code transparency is made possible by Zero Bit Insertion (ZBI).

The flag has a unique binary bit pattern: 01111110 (7E HEX). To eliminate the possibility of the data field containing a 7E HEX pattern, a bit stuffing technique

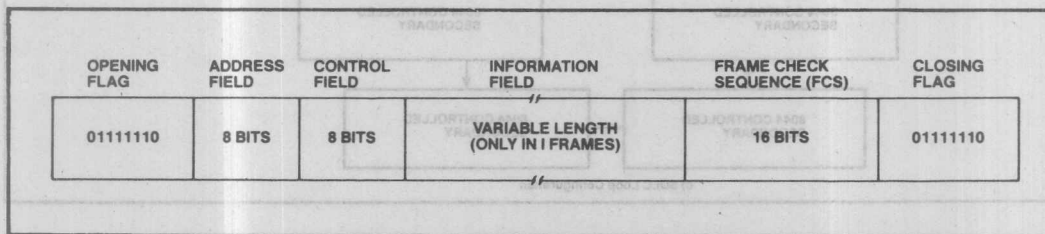


Figure 16-4. Frame Format

called Zero Bit Insertion is used. This technique specifies that during transmission, a binary 0 be inserted by the transmitter after any succession of five contiguous binary 1's. This will ensure that no pattern of 0 1 1 1 1 1 0 is ever transmitted between flags. On the receiving side, after receiving the flag, the receiver hardware automatically deletes any 0 following five consecutive 1's. The 8044 performs zero bit insertion and deletion automatically.

### 16.2.5 Non-return to Zero Inverted (NRZI)

NRZI is a method of clock and data encoding that is well suited to the HDLC/SDLC protocol. It allows HDLC/SDLC protocols to be used with low cost asynchronous modems. NRZI coding is done at the transmitter to enable clock recovery from the data at the receiver terminal by using standard digital phase locked loop (DPLL) techniques. NRZI coding specifies that the signal condition does not change for transmitting a 1, while an 0 causes a change of state. NRZI coding ensures that an active data line will have a transition at least every 5-bit times (recall Zero Bit Insertion), while contiguous 0's will cause a change of state. Thus, ZBI and NRZI encoding makes it possible for the 8044's on-chip DPLL to recover a receive clock (from received data) synchronized to the received data and at the same time ensure data transparency.

### 16.2.6 References

1. IBM Synchronous Data Link Control General Information GA27-3093-2 File No. GENL-09.
2. Standard Network Access Protocol Specification, DATAPAC Trans-Canada Telephone System CCG111.
3. IBM 3650 Retail Store System Loop Interface OEM Information, IBM, GA27-3098-0
4. Guidebook to Data Communications, Training Manual, Hewlett-Packard 5955-1715
5. "Serial Backplane Suits Multiprocessor Architectures", Mike Webb, *Computer Design*, July 1984, p. 85-96.
6. "Serial Bus Simplifies Distributed Control", P.D. MacWilliams, *Control Engineering*, June 1984, p. 101-104.
7. "Chips Support Two Local Area Networks", Bob Dahlberg, *Computer Design*, May 1984, p. 107-114.
8. "Build a VLSI-based Workstation for the Ethernet Environment", Mike Webb, *EDN*, 23 February 1984, p. 297-307.
9. "Networking With the 8044", Young Sohn & Charles Gopen, *Digital Design*, May 1984, p. 136-137.

## 16.3 RUP1™-44 DESIGN SUPPORT

### 16.3.1 Design Tool Support

A critical design consideration is time to market. Intel provides a sophisticated set of design tools to speed hardware and software development time of 8044 based products. These include ICE-44, ASM-51, PL/M-51, and EMV-44.

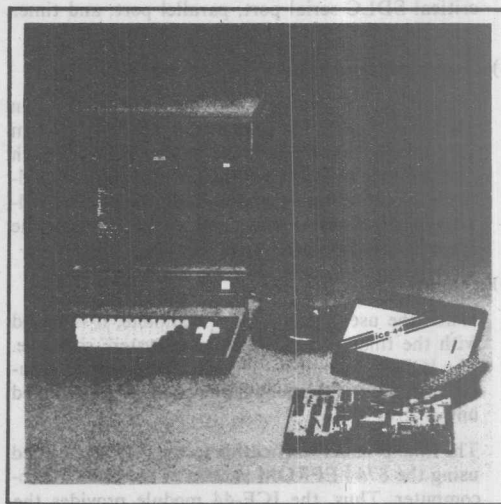


Figure 16-5. RUP1™-44 Development Support Configuration Intellect® System, ICE™-44 Buffer Box, and ICE-44 Module Plugged into a User Prototype Board.

A primary tool is the 8044 In Circuit Emulator, called ICE-44. See Figure 16-5. In conjunction with Intel's Intellect® Microprocessor Development System, the ICE-44 emulator allows hardware and software development to proceed interactively. This approach is more effective than the traditional method of independent hardware and software development followed by system integration. With the ICE-44 module, prototype hardware can be added to the system as it is designed. Software and hardware integration occurs while the product is being developed.

The ICE-44 emulator assists four stages of development:

#### 1) Software Debugging

It can be operated without being connected to the user's system before any of the user's hardware is available. In this stage ICE-44 debugging capabilities can be used in conjunction with the Intellect text edi-

tor and 8044 macroassembler to facilitate program development.

## 2) Hardware Development

The ICE-44 module's precise emulation characteristics and full-speed program RAM make it a valuable tool for debugging hardware, including the time-critical SDLC serial port, parallel port, and timer interfaces.

## 3) System Integration

Integration of software and hardware can begin when any functional element of the user system hardware is connected to the 8044 socket. As each section of the user's hardware is completed, it is added to the prototype. Thus, each section of the hardware and software is system tested in real-time operation as it becomes available.

## 4) System Test

When the user's prototype is complete, it is tested with the final version of the user system software. The ICE-44 module is then used for real-time emulation of the 8044 to debug the system as a completed unit.

The final product verification test may be performed using the 8744 EPROM version of the 8044 microcomputer. Thus, the ICE-44 module provides the user with the ability to debug a prototype or production system at any stage in its development.

A conversion kit, ICE-44 CON, is available to upgrade an ICE-51 module to ICE-44.

Intel's ASM-51 Assembler supports the 8044 special function registers and assembly program development. PL/M-51 provides designers with a high level language for the 8044. Programming in PL/M can greatly reduce development time, and ensure quick time to market.

These tools have recently been expanded with the addition of the EMV-44CON. This conversion kit allows you to convert an EMV-51 into an EMV-44 emulation vehicle. The resultant low cost emulator is design for use

with an iPDS Personal Development System, which also supports the ASM-51 assembler and PL/M-51. See Figure 16-6.



**Figure 16-6. RUP-44 iPDS Personal Development System, EMV-44 Buffer Box, and EMV-44 Module Plugged into a User Prototype Board.**

Emulation support is similar to the ICE-44 with support for Software and Hardware Development, System Integration, and System Test. The iPDS's rugged portability and ease of use also make it an ideal system for production tests and field service of your finished design. In addition, the iPDS offers EPROM programming module for the 8744, and direct communications with the 8044-based BITBUS via an optional iSBX344 distributed control module.

## 16.3.2 8051 Workshop

Intel provides 8051 training to its customers through the 5-day 8051 workshop. Familiarity with the 8051 and 8044 is achieved through a combination of lecture and laboratory exercises.

For designers not familiar with the 8051, the workshop is an effective way to become proficient with the 8051 architecture and capabilities.





17

8044 Architecture

## CHAPTER 17

### 8044 ARCHITECTURE

#### 17.0 GENERAL

The 8044 is based on the 8051 core. The 8044 replaces the 8051's serial port with an intelligent HDLC/SDLC controller called the Serial Interface or SIU. Thus the differences between the two result from the 8044's increased on-chip RAM (192 bytes) and additional special function registers necessary to control the SIU. Aside from the increased memory, the SIU itself, and differences in 5 pins (for the serial port), the 8044 and 8051 are compatible.

This chapter describes the differences between the 8044 and 8051. Information pertaining to the 8051 core, eg. instruction set, port operation, EPROM programming, etc. is located in the 8051 sections of this manual.

A block diagram of the 8044 is shown in Figure 17-1. The pinpoint is shown on the inside front cover.

#### 17.1 MEMORY ORGANIZATION OVERVIEW

The 8044 maintains separate address spaces for Program Memory and Data Memory. The Program Memory can be up to 64K bytes long, of which the lowest 4K bytes are in the on-chip ROM.

If the  $\overline{EA}$  pin is held high, the 8044 executes out of internal ROM unless the Program Counter exceeds 0FFFH. Fetches from locations 1000H through FFFFH are directed to external Program Memory.

If the  $\overline{EA}$  pin is held low, the 8044 fetches all instructions from external Program Memory.

The Data Memory consists of 192 bytes of on-chip RAM, plus 35 Special Function Registers, in addition to which the device is capable of accessing up to 64K bytes of external data memory.

The Program Memory uses 16-bit addresses. The external Data Memory can use either 8-bit or 16-bit addresses. The internal Data Memory uses 8-bit addresses, which provide a 256-location address space. The lower 192 addresses access the on-chip RAM. The Special Function Registers occupy various locations in the upper 128 bytes of the same address space.

The lowest 32 bytes in the internal RAM (locations 00 through 1FH) are divided into 4 banks of registers, each bank consisting of 8 bytes. Any one of these banks can be selected to be the "working registers" of the CPU, and can be accessed by a 3-bit address in the same byte as the opcode of an instruction. Thus, a large number of instructions are one-byte instructions.

The next higher 16 bytes of the internal RAM (locations 20H through 2FH) have individually addressable bits. These are provided for use as software flags or for one-bit (Boolean) processing. This bit-addressing capability is an important feature of the 8044. In addition to the 128 individually addressable bits in RAM, twelve of the Special Function Registers also have individually addressable bits.

A memory map is shown in Figure 17-2.

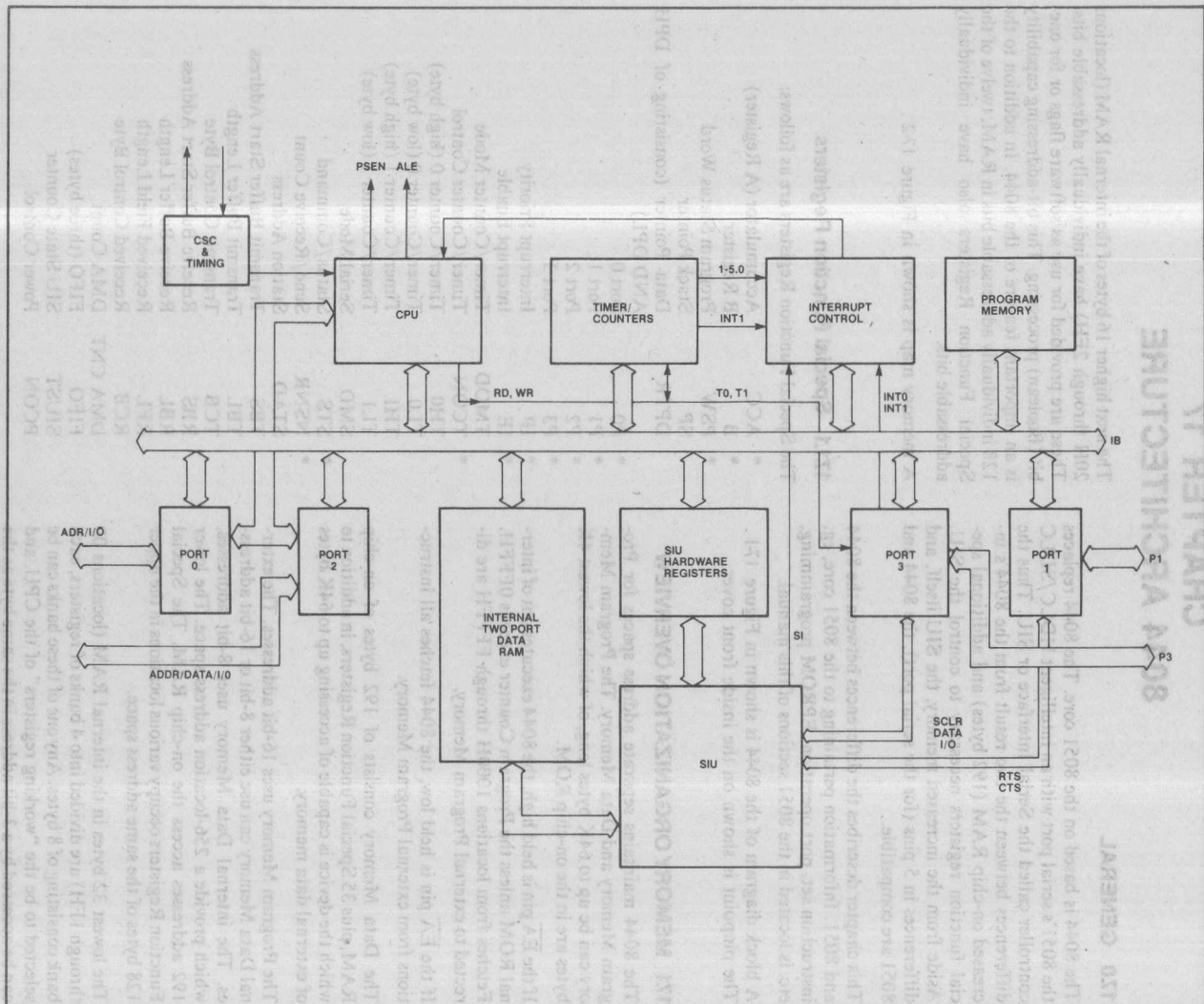
#### 17.1.1 Special Function Registers

The Special Function Registers are as follows:

* ACC	Accumulator (A Register)
* B	B Register
* PSW	Program Status Word
SP	Stack Pointer
DPTR	Data Pointer (consisting of DPH AND DPL)
* P0	Port 0
* P1	Port 1
* P2	Port 2
* P3	Port 3
* IP	Interrupt Priority
* IE	Interrupt Enable
TMOD	Timer/Counter Mode
* TCON	Timer/Counter Control
TH0	Timer/Counter 0 (high byte)
TL0	Timer/Counter 0 (low byte)
TH1	Timer/Counter 1 (high byte)
TL1	Timer/Counter 1 (low byte)
SMD	Serial Mode
* STS	Status/Command
* NSNR	Send/Receive Count
STAD	Station Address
TBS	Transmit Buffer Start Address
TBL	Transmit Buffer Length
TCB	Transmit Control Byte
RBS	Receive Buffer Start Address
RBL	Receive Buffer Length
RFL	Received Field Length
RCB	Received Control Byte
DMA CNT	DMA Count
FIFO	FIFO (three bytes)
SIUST	SIU State Counter
PCON	Power Control

The registers marked with \* are both byte- and bit-addressable.

Figure 17-1 RUP1™ Block Diagram



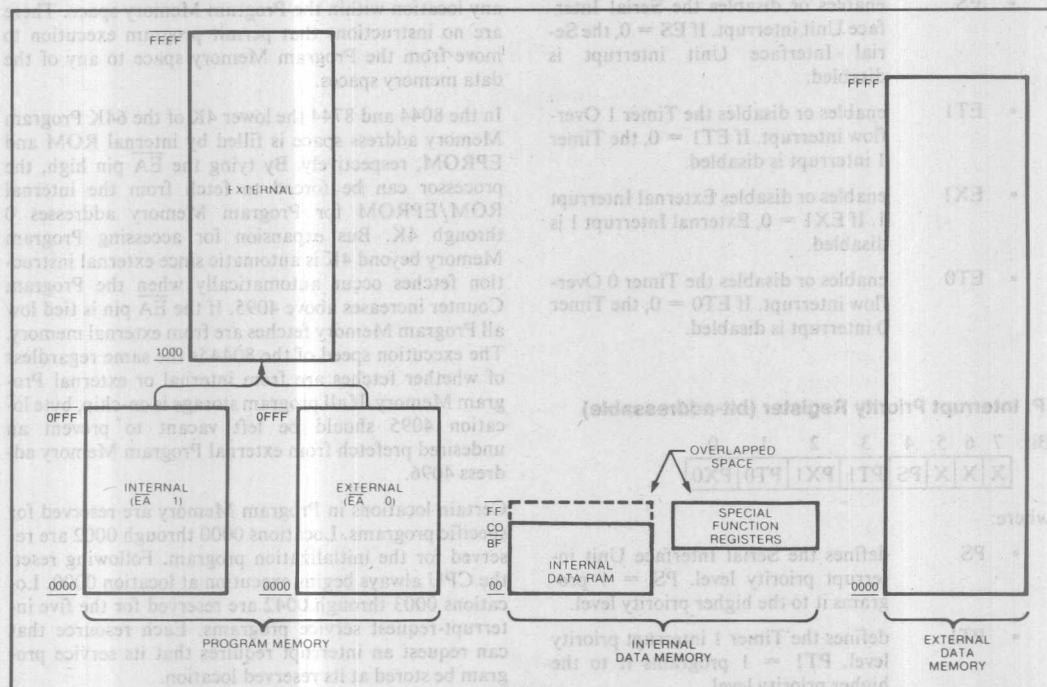


Figure 17-2. RUP1™-44 Memory Map

### Stack Pointer

The Stack Pointer is 8 bits wide. The stack can reside anywhere in the 192 bytes of on-chip RAM. When the 8044 is reset, the stack pointer is initialized to 07H. When executing a PUSH or a CALL, the stack pointer is incremented before data is stored, so the stack would begin at location 08H.

### 17.1.2 Interrupt Control Registers

The Interrupt Request Flags are as listed below:

Source	Request Flag	Location
External Interrupt 0	INT0, if IT0 = 0 IE0, if IT0 = 1	P3.2 TCON.1
Timer 0 Overflow	TF0	TCON.5
External Interrupt 1	INT1, if IT1 = 0 IE1, if IT1 = 1	P3.3 TCON.3
Timer 1 Overflow	TF1	TCON.7
Serial Interface Unit SI		STS.4

External Interrupt control bits IT0 and IT1 are in TCON.0 and TCON.2, respectively. Reset leaves all flags inactive, with IT0 and IT1 cleared.

All the interrupt flags can be set or cleared by software, with the same effect as by hardware.

The Enable and Priority Control Registers are shown below. All of these control bits are set or cleared by software. All are cleared by reset.

### IE: Interrupt Enable Register (bit-addressable)

Bit:	7	6	5	4	3	2	1	0
	EA	X	X	ES	ET1	EX1	ET0	EX0

where:

EA disables all interrupts. If EA = 0, no interrupt will be acknowledged. If EA = 1, each interrupt source is individually enabled or disabled by setting or clearing its enable bit.



- **ES** enables or disables the Serial Interface Unit interrupt. If ES = 0, the Serial Interface Unit interrupt is disabled.
- **ET1** enables or disables the Timer 1 Overflow interrupt. If ET1 = 0, the Timer 1 interrupt is disabled.
- **EX1** enables or disables External Interrupt 1. If EX1 = 0, External Interrupt 1 is disabled.
- **ET0** enables or disables the Timer 0 Overflow interrupt. If ET0 = 0, the Timer 0 interrupt is disabled.

#### IP: Interrupt Priority Register (bit-addressable)

Bit:	7	6	5	4	3	2	1	0
	X	X	X	PS	PT1	PX1	PT0	PX0

where:

- **PS** defines the Serial Interface Unit interrupt priority level. PS = 1 programs it to the higher priority level.
- **PT1** defines the Timer 1 interrupt priority level. PT1 = 1 programs it to the higher priority level.
- **PX1** defines the External Interrupt 1 priority level. PX1 = 1 programs it to the higher priority level.
- **PT0** defines the Timer 0 interrupt priority level. PT0 = 1 programs it to the higher priority level.
- **PX0** defines the External Interrupt 0 priority level. PX0 = 1 programs it to the higher priority level.

### 17.2 Memory Organization Details

In the 8044 family the memory is organized over three address spaces and the program counter. The memory spaces shown in Figure 18-2 are the:

- 64K-byte Program Memory address space
- 64K-byte External Data Memory address space
- 320-byte Internal Data Memory address space

The 16-bit Program Counter register provides the 8044 with its 64K addressing capabilities. The Program Counter allows the user to execute calls and branches to

any location within the Program Memory space. There are no instructions that permit program execution to move from the Program Memory space to any of the data memory spaces.

In the 8044 and 8744 the lower 4K of the 64K Program Memory address space is filled by internal ROM and EPROM, respectively. By tying the EA pin high, the processor can be forced to fetch from the internal ROM/EPROM for Program Memory addresses 0 through 4K. Bus expansion for accessing Program Memory beyond 4K is automatic since external instruction fetches occur automatically when the Program Counter increases above 4095. If the EA pin is tied low all Program Memory fetches are from external memory. The execution speed of the 8044 is the same regardless of whether fetches are from internal or external Program Memory. If all program storage is on-chip, byte location 4095 should be left vacant to prevent an undesired prefetch from external Program Memory address 4096.

Certain locations in Program Memory are reserved for specific programs. Locations 0000 through 0002 are reserved for the initialization program. Following reset, the CPU always begins execution at location 0000. Locations 0003 through 0042 are reserved for the five interrupt-request service programs. Each resource that can request an interrupt requires that its service program be stored at its reserved location.

The 64K-byte External Data Memory address space is automatically accessed when the MOVX instruction is executed.

Functionally the Internal Data Memory is the most flexible of the address spaces. The Internal Data Memory space is subdivided into a 256-byte Internal Data RAM address space and a 128-byte Special Function Register address space as shown in Figure 17-3.

The Internal Data RAM address space is 0 to 255. Four 8-Register Banks occupy locations 0 through 31. The stack can be located anywhere in the Internal Data RAM address space. In addition, 128 bit locations of the on-chip RAM are accessible through Direct Addressing. These bits reside in Internal Data RAM at byte locations 32 through 47. Currently locations 0 through 191 of the Internal Data RAM address space are filled with on-chip RAM.

The stack depth is limited only by the available Internal Data RAM, thanks to an 8-bit reloadable Stack Pointer. The stack is used for storing the Program Counter during subroutine calls and may be used for passing parameters. Any byte of Internal Data RAM or Special Function Register accessible through Direct Addressing can be pushed/popped.

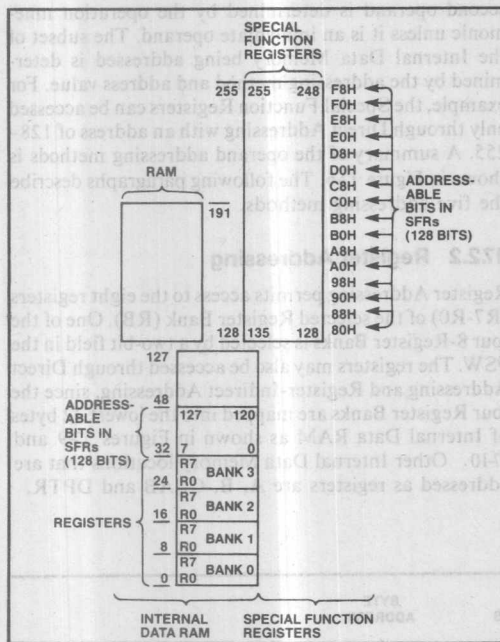


Figure 17-3. Internal Data Memory Address Space

The Special Function Register address space is 128 to 255. All registers except the Program Counter and the four 8-Register Banks reside here. Memory mapping the Special Function Registers allows them to be accessed as easily as internal RAM. As such, they can be operated on by most instructions. In the overlapping memory space (address 128-191), indirect addressing is used to access RAM, and direct addressing is used to access the SFR's. The SFR's at addresses 192-255 are also accessed using direct addressing. The Special Function Registers are listed in Figure 17-4. Their mapping in the Special Function Register address space is shown in Figures 17-5 and 17-6.

Performing a read from a location of the Internal Data memory where neither a byte of Internal Data RAM (i.e., RAM addresses 192-255) nor a Special Function Register exists will access data of indeterminable value.

Architecturally, each memory space is a linear sequence of 8-bit wide bytes. By Intel convention the storage of multi-byte address and data operands in program and data memories is the least significant byte at the low-order address and the most significant byte at the high-order address. Within byte X, the most significant bit is

**ARITHMETIC REGISTERS:**  
Accumulator\*, B register\*,  
Program Status Word\*

**POINTERS:**  
Stack Pointer, Data Pointer (high & low)

**PARALLEL I/O PORTS:**  
Port 3\*, Port 2\*, Port 1\*, Port 0\*

**INTERRUPT SYSTEM:**  
Interrupt Priority Control\*,  
Interrupt Enable Control\*

**TIMERS:**  
Timer MODE, Timer CONTROL\*, Timer 1  
(high & low), Timer 0 (high & low)

**SERIAL INTERFACE UNIT:**  
Transmit Buffer Start,  
Transmit Buffer Length,  
Transmit Control Byte,  
Send Count Receive Count\*,  
DMA Count,  
Station Address  
Receive Field Length  
Receive Buffer Start  
Receive Buffer Length  
Receive Control Byte,  
Serial Mode,  
Status Register.\*

\* Bits in these registers are bit addressable.

Figure 17-4. Special Function Registers

represented by X.7 while the least significant bit is X.0. Any deviation from these conventions will be explicitly stated in the text.

### 17.2.1 Operand Addressing

There are five methods of addressing source operands. They are Register Addressing, Direct Addressing, Register-Indirect Addressing, Immediate Addressing and Base-Register-plus Index-Register-Indirect Addressing. The first three of these methods can also be used to address a destination operand. Since operations in the 8044 require 0 (NOP only), 1, 2, 3 or 4 operands, these five addressing methods are used in combinations to provide the 8044 with its 21 addressing modes.

Most instructions have a "destination, source" field that specifies the data type, addressing methods and operands involved. For operations other than moves, the destination operand is also a source operand. For example, in "subtract-with-borrow A, #5" the A register receives the result of the value in register A minus 5, minus C.

Most operations involve operands that are located in Internal Data Memory. The selection of the Program Memory space or External Data Memory space for a

ARITHMETIC REGISTERS:  
Accumulator\*, B register\*,  
Program Status Word\*

POINTERS:  
Stack Pointer, Data Pointer (high & low)

PARALLEL I/O PORTS:  
Port 3\*, Port 2\*, Port 1\*, Port 0\*

INTERRUPT SYSTEM:  
Interrupt Priority Control\*,  
Interrupt Enable Control\*

TIMERS:  
Timer Mode, Timer Control\*, Timer 1 (high & low), Timer 0 (high & low)

SERIAL INTERFACE UNIT:  
Serial Mode, Status/Command\*,  
Send/Receive Count\*, Station Address,  
Transmit Buffer Start Address,  
Transmit Buffer Length,  
Transmit Control Byte,  
Receive Buffer Start Address,  
Receive Buffer Length,  
Receive Field Length,  
Receive Control Byte,  
DMA Count,  
FIFO (three bytes),  
SIU Controller State Counter

\* Bits in these registers are bit-addressable

Figure 17-5. Mapping of Special Function Registers

second operand is determined by the operation mnemonic unless it is an immediate operand. The subset of the Internal Data Memory being addressed is determined by the addressing method and address value. For example, the Special Function Registers can be accessed only through Direct Addressing with an address of 128-255. A summary of the operand addressing methods is shown in Figure 17-6. The following paragraphs describe the five addressing methods.

## 17.2.2 Register Addressing

Register Addressing permits access to the eight registers (R7-R0) of the selected Register Bank (RB). One of the four 8-Register Banks is selected by a two-bit field in the PSW. The registers may also be accessed through Direct Addressing and Register-Indirect Addressing, since the four Register Banks are mapped into the lowest 32 bytes of Internal Data RAM as shown in Figures 17-9 and 17-10. Other Internal Data Memory locations that are addressed as registers are A, B, C, AB and DPTR.

REGISTER NAMES	SYMBOLIC ADDRESS	BIT ADDRESS	BYTE ADDRESS
B REGISTER	B	247 through 240	240 (F0H) ←
ACCUMULATOR	ACC	231 through 224	224 (E0H) ←
*THREE BYTE FIFO	FIFO		223 (DFH) ←
	FIFO		222 (DEH) ←
	FIFO		221 (DDH) ←
TRANSMIT BUFFER START	TBS		220 (DCH) ←
TRANSMIT BUFFER LENGTH	TBL		219 (DBH) ←
TRANSMIT CONTROL BYTE	TCB		218 (DAH) ←
*SIU STATE COUNTER	SIUST		217 (D9H) ←
SEND COUNT RECEIVE COUNT	NSNR	223 through 216	216 (D8H) ←
PROGRAM STATUS WORD	PSW	215 through 208	208 (D0H) ←
*DMA COUNT	DMA CNT		207 (CFH) ←
STATION ADDRESS	STAD		206 (CEH) ←
RECEIVE FIELD LENGTH	RFL		205 (CDH) ←
RECEIVE BUFFER START	RBS		204 (CCH) ←
RECEIVE BUFFER LENGTH	RBL		203 (CBH) ←
RECEIVE CONTROL BYTE	RCB		202 (CAH) ←
SERIAL MODE	SMD		201 (C9H) ←
STATUS REGISTER	STS	207 through 200	200 (C8H) ←
INTERRUPT PRIORITY CONTROL	IP		184 (B8H) ←
PORT 3	P3	191 through 184	176 (B0H) ←
INTERRUPT ENABLE CONTROL	IE	175 through 168	168 (A8H) ←
PORT 2	P2	167 through 160	160 (A0H) ←
PORT 1	P1	151 through 144	144 (90H) ←
TIMER HIGH 1	TH1		141 (8DH) ←
TIMER HIGH 0	TH0		140 (8CH) ←
TIMER LOW 1	TL1		139 (8BH) ←
TIMER LOW 0	TL0		138 (8AH) ←
TIMER MODE	TMOD		137 (89H) ←
TIMER CONTROL	TCON	143 through 136	136 (88H) ←
DATA POINTER HIGH	DPH		131 (83H) ←
DATA POINTER LOW	DPL		130 (82H) ←
STACK POINTER	SP		129 (81H) ←
PORT 0	P0	135 through 128	128 (80H) ←

\*ICE Support Hardware registers. Under normal operating conditions there is no need for the CPU to access these registers.

Figure 17-6. Mapping of Special Function Registers

Direct Byte Address (MSB)	Bit Address	Hardware Register Symbol
240	F7 F6 F5 F4 F3 F2 F1 F0	8
224	E7 E6 E5 E4 E3 E2 E1 E0	ACC
216	NS2 NS1 NS0 SES NR2 NR1 NR0 SER	NSNR
204	D7 D6 D5 D4 D3 D2 D1 D0	PSW
200	TBF RE RTS SI BV CPB AM RBP	STS
184	CF CE CD CC CB CA C9 C8	1P
	PS PT1 PX1 PT0 PX0	
176	B7 B6 B5 B4 B3 B2 B1 B0	P3
168	EA E5 ET1 EX1 ET0 EX0	1E
160	AF — — AC AB AA A9 A8	
144	A7 A6 A5 A4 A3 A2 A1 A0	P2
136	97 96 95 94 93 92 91 90	P1
128	TF1 TR1 TF0 TR0 IE1 IT1 IE0 IT0	TCN
	8F 8E 8D 8C 8B 8A 89 88	
	87 86 85 84 83 82 81 80	P0

Figure 17-7. Special Function Register Bit Address

RAM BYTE (MSB)	Bit Address (LSB)
8FH	191
7FH	7E
7EH	7D
7DH	7C
7CH	7B
7BH	7A
7AH	79
79H	78
78H	77
77H	76
76H	75
75H	74
74H	73
73H	72
72H	71
71H	70
70H	6F
6FH	6E
6EH	6D
6DH	6C
6CH	6B
6BH	6A
6AH	69
69H	68
68H	67
67H	66
66H	65
65H	64
64H	63
63H	62
62H	61
61H	60
60H	5F
5FH	5E
5EH	5D
5DH	5C
5CH	5B
5BH	5A
5AH	59
59H	58
58H	57
57H	56
56H	55
55H	54
54H	53
53H	52
52H	51
51H	50
50H	4F
4FH	4E
4EH	4D
4DH	4C
4CH	4B
4BH	4A
4AH	49
49H	48
48H	47
47H	46
46H	45
45H	44
44H	43
43H	42
42H	41
41H	40
40H	3F
3FH	3E
3EH	3D
3DH	3C
3CH	3B
3BH	3A
3AH	39
39H	38
38H	37
37H	36
36H	35
35H	34
34H	33
33H	32
32H	31
31H	30
30H	2F
2FH	2E
2EH	2D
2DH	2C
2CH	2B
2BH	2A
2AH	29
29H	28
28H	27
27H	26
26H	25
25H	24
24H	23
23H	22
22H	21
21H	20
20H	1F
1FH	1E
1EH	1D
1DH	1C
1CH	1B
1BH	1A
1AH	19
19H	18
18H	17
17H	16
16H	15
15H	14
14H	13
13H	12
12H	11
11H	10
10H	0F
0FH	0E
0EH	0D
0DH	0C
0CH	0B
0BH	0A
0AH	09
09H	08
08H	07
07H	06
06H	05
05H	04
04H	03
03H	02
02H	01
01H	00
00H	0

Figure 17-9. RAM Bit Addresses

<ul style="list-style-type: none"> <li>Register Addressing <ul style="list-style-type: none"> <li>R7-R0 <ul style="list-style-type: none"> <li>A,B,C (bit), AB (two bytes), DPTR (double byte)</li> </ul> </li> </ul> </li> <li>Direct Addressing <ul style="list-style-type: none"> <li>Lower 128 bytes of Internal Data RAM</li> <li>Special Function Registers <ul style="list-style-type: none"> <li>128 bits in subset of Special Function Register address space</li> </ul> </li> </ul> </li> <li>Register-Indirect Addressing <ul style="list-style-type: none"> <li>Internal Data RAM (@R1, @R0, @SP (PUSH and POP only))</li> <li>Least Significant Nibbles in Internal Data RAM (@R1, @R0)</li> <li>External Data Memory (@R1, @R0, @DPTR)</li> </ul> </li> <li>Immediate Addressing <ul style="list-style-type: none"> <li>Program Memory (in-code constant)</li> </ul> </li> <li>Base-Register-plus Index-Register-Indirect Addressing <ul style="list-style-type: none"> <li>Program Memory (@ DPTR + A, @ PC+A)</li> </ul> </li> </ul>
--

Figure 17-8. Operand Addressing Methods

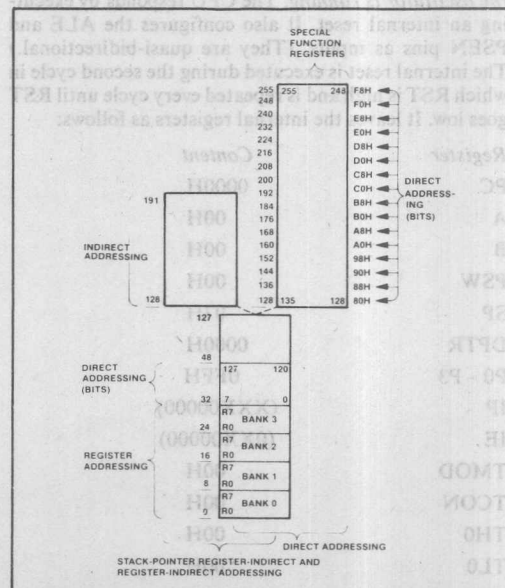


Figure 17-10. Addressing Operands in Internal Data Memory



### 17.2.3 Direct Addressing

Direct Addressing provides the only means of accessing the memory-mapped byte-wide Special Function Registers and memory mapped bits within the Special Function Registers and Internal Data RAM. Direct Addressing of bytes may also be used to access the lower 128 bytes of Internal Data RAM. Direct Addressing of bits gains access to a 128-bit subset of the Internal Data RAM and 128 bit subset of the Special Function Registers as shown in Figures 17-5, 17-6, 17-9, and 17-10.

Register-Indirect Addressing using the content of R1 or R0 in the selected Register Bank, or using the content of the Stack Pointer (PUSH and POP only), addresses the Internal Data RAM. Register-Indirect Addressing is also used for accessing the External Data Memory. In this case, either R1 or R0 in the selected Register Bank may be used for accessing locations within a 256-byte block. The block number can be preselected by the contents of a port. The 16-bit Data Pointer may be used for accessing any location within the full 64K external address space.

### 17.3 RESET

Reset is accomplished by holding the RST pin high for at least two machine cycles (24 oscillator periods) while the oscillator is running. The CPU responds by executing an internal reset. It also configures the ALE and PSEN pins as inputs. (They are quasi-bidirectional.) The internal reset is executed during the second cycle in which RST is high and is repeated every cycle until RST goes low. It leaves the internal registers as follows:

Register	Content
PC	0000H
A	00H
B	00H
PSW	00H
SP	07H
DPTR	0000H
P0 - P3	0FFH
IP	(XXX00000)
IE	(0XX00000)
TMOD	00H
TCON	00H
TH0	00H
TL0	00H
TH1	00H
TL1	00H

SMD	00H
STS	00H
NSNR	00H
STAD	00H
TBS	00H
TBL	00H
TCB	00H
RBS	00H
RBL	00H
RFL	00H
RCB	00H
DMA CNT	00H
FIFO1	00H
FIFO2	00H
FIFO3	00H
SIUST	01H
PCON	(0XXXXXXX)

The internal RAM is not affected by reset. When VCC is turned on, the RAM content is indeterminate unless VPD was applied prior to VCC being turned off (see Power Down Operation.)

### 17.4 RUPI™-44 FAMILY PIN DESCRIPTION

**VSS:** Circuit ground potential.

**VCC:** Supply voltage during programming (of the 8744), verification (of the 8044 or 8744), and normal operation.

**Port 0:** Port 0 is an 8-bit open drain bidirectional I/O port. It is also the multiplexed low-order address and data bus during accesses to external memory (during which accesses it activates internal pullups). It also outputs instruction bytes during program verification. (External pullups are required during program verification.) Port 0 can sink eight LS TTL inputs.

**Port 1:** Port 1 is an 8-bit bidirectional I/O port with internal pullups. It receives the low-order address byte during program verification in the 8044 or 8744. Port 1 can sink/source four LS TTL inputs. It can drive MOS inputs without external pullups.

Two of the Port 1 pins serve alternate functions, as listed below:

Port Pin	Alternate Function
P1.6	RTS (Request to Send). In a non-loop configuration, RTS signals that the 8044 is ready to transmit data.

**P1.7**  $\overline{\text{CTS}}$  (Clear to Send). In a non-loop configuration,  $\overline{\text{CTS}}$  signals to the 8044 that the receiving station is ready to accept data.

**Port 2:** Port 2 is an 8-bit bidirectional I/O port with internal pullups. It emits the high-order address byte during accesses to external memory. It also receives the high-order address bits and control signals during program verification in the 8044 or 8744. Port 2 can sink/source four LS TTL inputs. It can drive MOS inputs without external pullups.

**Port 3:** Port 3 is an 8-bit bidirectional I/O port with internal pullups. Port 3 can sink/source four LS TTL inputs. It can drive MOS inputs without external pullups.

Port 3 pins also serve alternate functions, as listed below:

Port Pin	Alternate Function
P3.0	RXD (serial input port in loop configuration). I/O (data direction control in non-loop configuration).
P3.1	TXD (serial output port in loop configuration). DATA input/output pin in non-loop configuration.
P3.2	$\overline{\text{INT0}}$ (external interrupt)
P3.3	$\overline{\text{INT1}}$ (external interrupt)
P3.4	T0 (Timer 0 external input)
P3.5	T1 (Timer 1 external input) SCLK (Serial Data Clock Input)
P3.6	$\overline{\text{WR}}$ (external Data Memory write strobe)
P3.7	$\overline{\text{RD}}$ (external Data Memory read strobe)

**RST/VPD:** A high level on this pin for two machine cycles while the oscillator is running resets the device. An

internal pulldown permits Power-On reset using only a capacitor connected to VCC.

**ALE/PROG:** Address Latch Enable output for latching the low byte of the address during accesses to external memory. ALE is activated though for this purpose at a constant rate of 1/6 the oscillator frequency even when external memory is not being accessed. Consequently it can be used for external clocking or timing purposes. (However, one ALE pulse is skipped during each access to external Data Memory.) This pin is also the program pulse input (PROG) during EPROM programming.

**PSEN:** Program Store Enable output is the read strobe to external Program Memory. PSEN is activated twice each machine cycle during fetches from external Program Memory. (However, when executing out of external Program Memory two activations of PSEN are skipped during each access to external Data Memory.) PSEN is not activated during fetches from internal Program Memory.

**EA/VPP:** When EA is held high the CPU executes out of internal Program Memory (unless the Program Counter exceeds 0FFFFH). When EA is held low the CPU executes only out of external Program Memory. In the 8344, EA must be externally wired low. In the 8744, this pin also receives the 21V programming supply voltage (VPP) during EPROM programming.

**XTAL1:** Input to the inverting amplifier that forms the oscillator. Should be grounded when an external oscillator is used.

**XTAL2:** Output of the inverting amplifier that forms the oscillator, and input to the internal clock generator. Receives the external oscillator signal when an external oscillator is used.



---





## CHAPTER 18

# THE 8044 SERIAL INTERFACE UNIT

### 18.0 SERIAL INTERFACE

The serial interface provides a high-performance communication link. The protocol used for this communication is based on the IBM Synchronous Data Link Control (SDLC). The serial interface also supports a subset of the ISO HDLC (International Standards Organization High-Level Data Link Control) protocol.

The SDLC/HDLC protocols have been accepted as standard protocols for many high-level teleprocessing systems. The serial interface performs many of the functions required to service the data link without intervention from the 8044's own CPU. The programmer is free to concentrate on the 8044's function as a peripheral controller, rather than having to deal with the details of the communication process.

Five pins on the 8044 are involved with the serial interface (refer to Section 12.4, Family Pin Description, for details):

Pin 7	$\overline{\text{RTS}}/\text{P16}$
Pin 8	$\overline{\text{CTS}}/\text{P17}$
Pin 10	$\text{I}/\overline{\text{O}}/\text{RXD}/\text{P30}$
Pin 11	$\text{DATA}/\text{TXD}/\text{P31}$
Pin 15	$\text{SCLK}/\text{T1}/\text{P35}$

Figure 18-1 is a functional block diagram of the serial interface unit (SIU). More details on the SIU hardware are given in Section 18.9.

### 18.1 DATA LINK CONFIGURATIONS

The serial interface is capable of operating in three serial data link configurations:

- 1) Half-Duplex, point-to-point
- 2) Half-Duplex, multipoint (with a half-duplex or full-duplex primary)
- 3) Loop

Figure 18-2 shows these three configurations. The RTS (Request to Send) and CTS (Clear to Send) handshaking signals are available in the point-to-point and multipoint configurations.

### 18.2 DATA CLOCKING OPTIONS

The serial interface can operate in an externally clocked mode or in a self clocked mode.

#### Externally Clocked Mode

In the externally clocked mode, a common Serial Data Clock (SCLK on pin 15) synchronizes the serial bit stream. This clock signal may come from the master CPU or primary station, or from an external phase-locked loop local to the 8044. Figure 18-3 illustrates the timing relationships for the serial interface signals when the externally clocked mode is used in point-to-point and multipoint data link configurations.

Incoming data is sampled at the rising edge of SCLK, and outgoing data is shifted out at the falling edge of SCLK. More detailed timing information is given in the 8044 data sheet.

#### Self Clocked (Asynchronous) Mode

The self clocked mode allows data transfer without a common system data clock. Using an on-chip DPLL (digital phase locked loop) the serial interface recovers the data clock from the data stream itself. The DPLL requires a reference clock equal to either 16 times or 32 times the data rate. This reference clock may be externally supplied or internally generated. When the serial interface generates this clock internally, it uses either the 8044's internal logic clock (half the crystal frequency's PH2) or the "timer 1" overflow. Figure 18-4 shows the serial interface signal timing relationships for the loop configuration, when the unclocked mode is used.

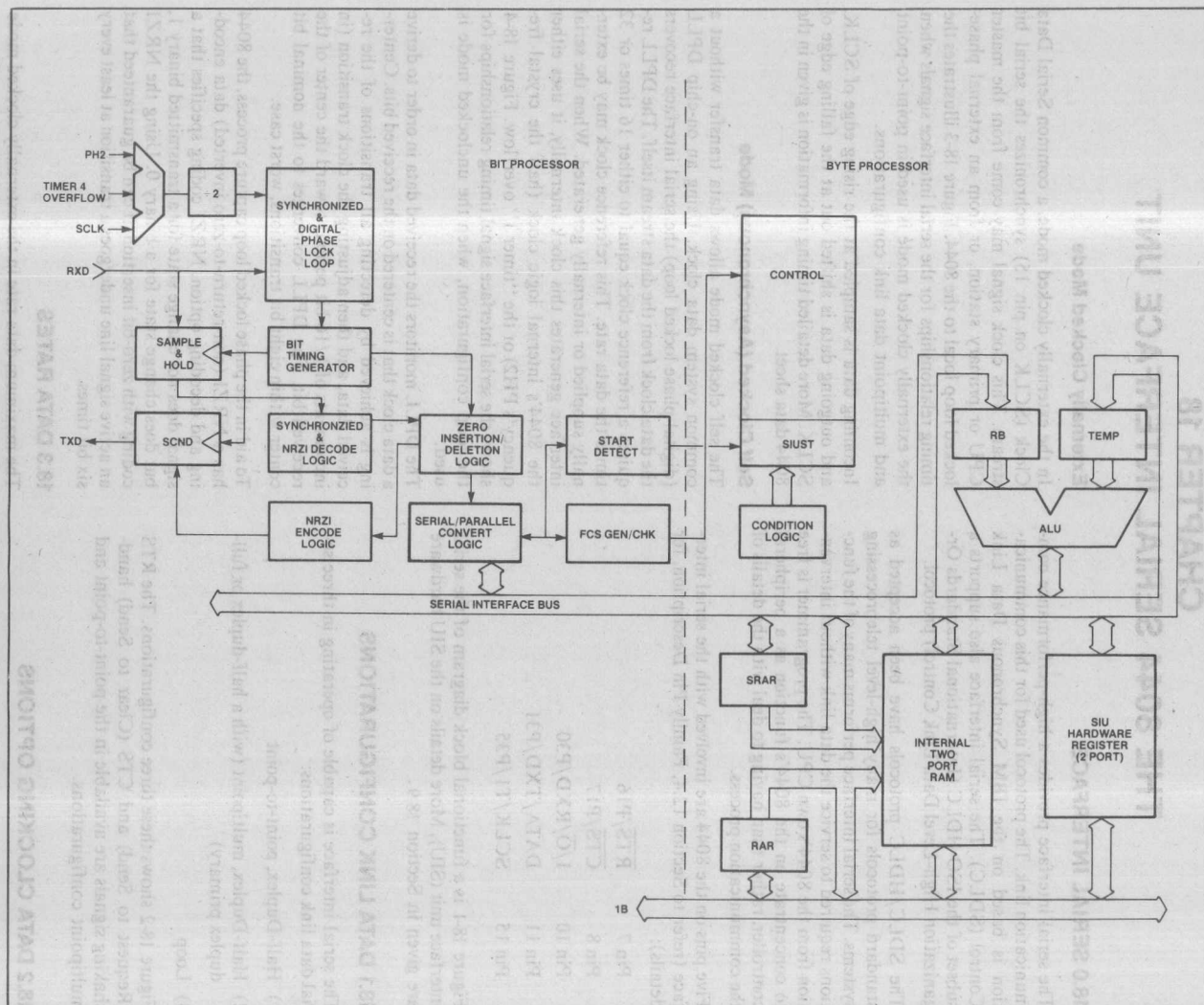
The DPLL monitors the received data in order to derive a data clock that is centered on the received bits. Centering is achieved by detecting all transitions of the received data, and then adjusting the clock transition (in increments of 1/16 bit period) toward the center of the received bit. The DPLL converges to the nominal bit center within eight bit transitions, worst case.

To aid in the phase locked loop capture process, the 8044 has a NRZI (non-return-to-zero inverted) data encoding and decoding option. NRZI coding specifies that a signal does not change state for a transmitted binary 1, but does change state for a binary 0. Using the NRZI coding with zero-bit insertion, it can be guaranteed that an active signal line undergoes a transition at least every six bit times.

### 18.3 DATA RATES

The maximum data rate in the externally clocked mode is 2.4M bits per second (bps) a half-duplex configuration, and 1.0M in a loop configuration.

Figure 18-1. SIU Block Diagram



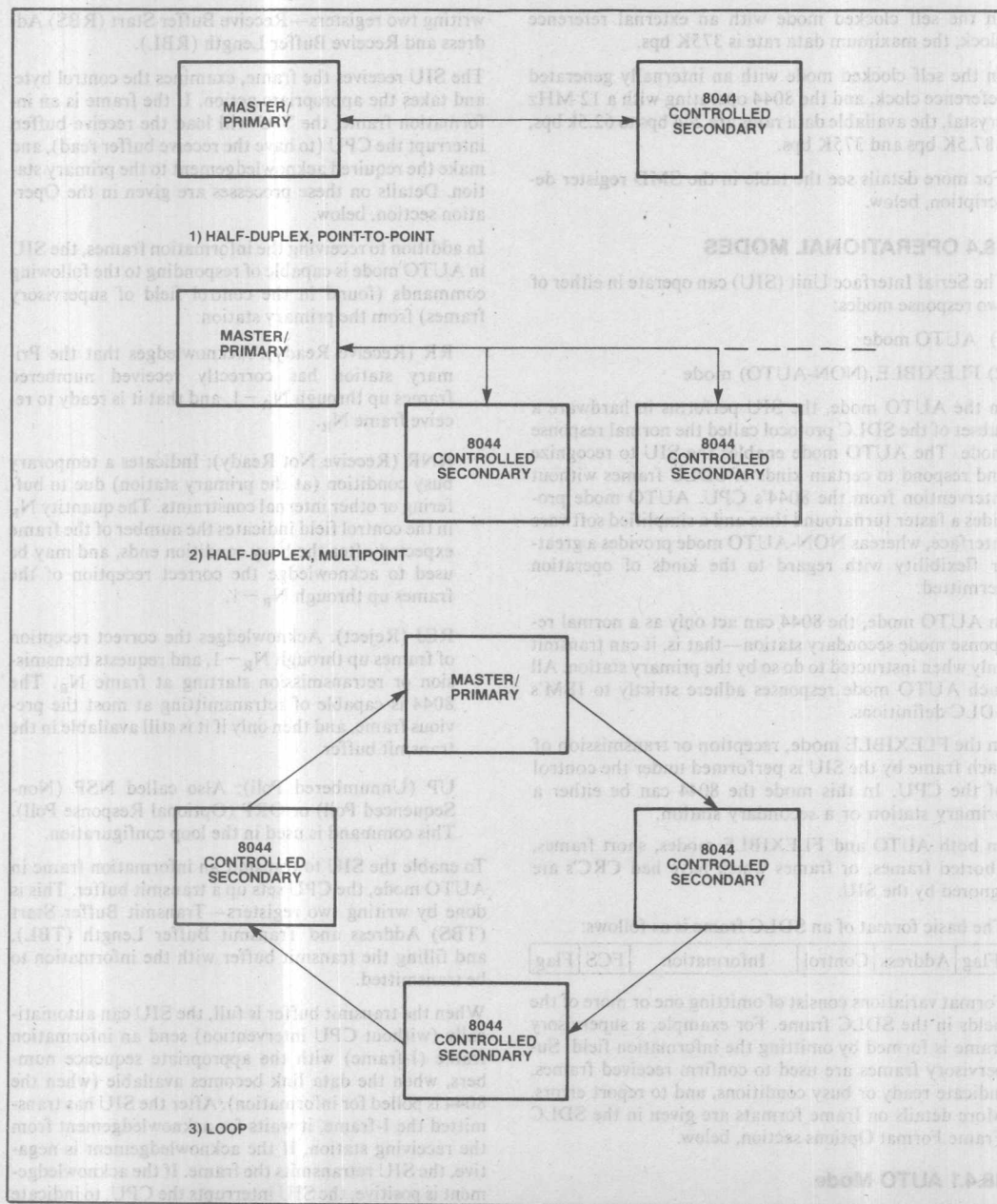


Figure 18-2 . RUPI-44 Data Link Configurations



In the self clocked mode with an external reference clock, the maximum data rate is 375K bps.

In the self clocked mode with an internally generated reference clock, and the 8044 operating with a 12 MHz crystal, the available data rates are 244 bps to 62.5k bps, 187.5K bps and 375K bps.

For more details see the table in the SMD register description, below.

## 18.4 OPERATIONAL MODES

The Serial Interface Unit (SIU) can operate in either of two response modes:

- 1) AUTO mode
- 2) FLEXIBLE (NON-AUTO) mode

In the AUTO mode, the SIU performs in hardware a subset of the SDLC protocol called the normal response mode. The AUTO mode enables the SIU to recognize and respond to certain kinds of SDLC frames without intervention from the 8044's CPU. AUTO mode provides a faster turnaround time and a simplified software interface, whereas NON-AUTO mode provides a greater flexibility with regard to the kinds of operation permitted.

In AUTO mode, the 8044 can act only as a normal response mode secondary station—that is, it can transmit only when instructed to do so by the primary station. All such AUTO mode responses adhere strictly to IBM's SDLC definitions.

In the FLEXIBLE mode, reception or transmission of each frame by the SIU is performed under the control of the CPU. In this mode the 8044 can be either a primary station or a secondary station.

In both AUTO and FLEXIBLE modes, short frames, aborted frames, or frames which have had CRC's are ignored by the SIU.

The basic format of an SDLC frame is as follows:

Flag	Address	Control	Information	FCS	Flag
------	---------	---------	-------------	-----	------

Format variations consist of omitting one or more of the fields in the SDLC frame. For example, a supervisory frame is formed by omitting the information field. Supervisory frames are used to confirm received frames, indicate ready or busy conditions, and to report errors. More details on frame formats are given in the SDLC Frame Format Options section, below.

### 18.4.1 AUTO Mode

To enable the SIU to receive a frame in AUTO mode, the 8044 CPU sets up a receive buffer. This is done by

writing two registers—Receive Buffer Start (RBS) Address and Receive Buffer Length (RBL).

The SIU receives the frame, examines the control byte, and takes the appropriate action. If the frame is an information frame, the SIU will load the receive buffer, interrupt the CPU (to have the receive buffer read), and make the required acknowledgement to the primary station. Details on these processes are given in the Operation section, below.

In addition to receiving the information frames, the SIU in AUTO mode is capable of responding to the following commands (found in the control field of supervisory frames) from the primary station:

**RR (Receive Ready):** Acknowledges that the Primary station has correctly received numbered frames up through  $N_R - 1$ , and that it is ready to receive frame  $N_R$ .

**RNR (Receive Not Ready):** Indicates a temporary busy condition (at the primary station) due to buffering or other internal constraints. The quantity  $N_R$  in the control field indicates the number of the frame expected after the busy condition ends, and may be used to acknowledge the correct reception of the frames up through  $N_R - 1$ .

**REJ (Reject):** Acknowledges the correct reception of frames up through  $N_R - 1$ , and requests transmission or retransmission starting at frame  $N_R$ . The 8044 is capable of retransmitting at most the previous frame, and then only if it is still available in the transmit buffer.

**UP (Unnumbered Poll):** Also called NSP (Non-Sequenced Poll) or ORP (Optional Response Poll). This command is used in the loop configuration.

To enable the SIU to transmit an information frame in AUTO mode, the CPU sets up a transmit buffer. This is done by writing two registers—Transmit Buffer Start (TBS) Address and Transmit Buffer Length (TBL), and filling the transmit buffer with the information to be transmitted.

When the transmit buffer is full, the SIU can automatically (without CPU intervention) send an information frame (I-frame) with the appropriate sequence numbers, when the data link becomes available (when the 8044 is polled for information). After the SIU has transmitted the I-frame, it waits for acknowledgement from the receiving station. If the acknowledgement is negative, the SIU retransmits the frame. If the acknowledgement is positive, the SIU interrupts the CPU, to indicate that the transmit buffer may be reloaded with new information.

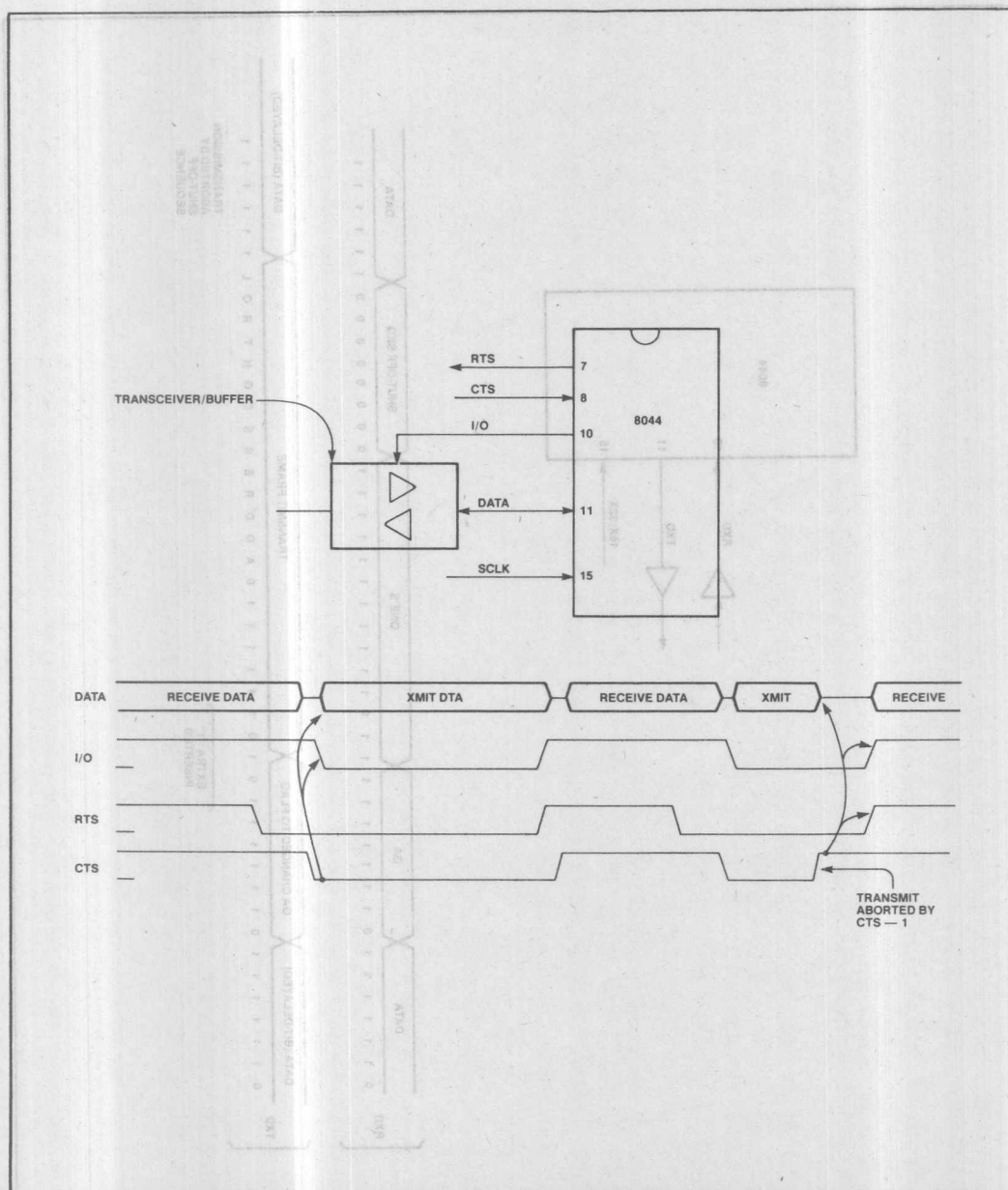


Figure 18-3. Serial Interface Timing—Clocked Mode



In addition to transmitting the information frames, the SIU in AUTO mode is capable of sending the following responses to the primary station:

**RR (Receive Ready):** Acknowledges that the 8044 has correctly received numbered frames up through  $N_R - 1$ , and that it is ready to receive frame  $N_R$ .

**RNR (Receive Not Ready):** Indicates a temporary busy condition (at the 8044) due to buffering or other internal constraints. The quantity  $N_R$  in the control field indicates the number of the frame expected after the busy condition ends, and acknowledges the correct reception of the frames up through  $N_R - 1$ .

#### 18.4.2 FLEXIBLE Mode

In the FLEXIBLE (or non-auto) mode, all reception and transmission is under the control of the CPU. The full SDLC and HDLC protocols can be implemented, as well as any bit-synchronous variants of these protocols.

FLEXIBLE mode provides more flexibility than AUTO mode, but it requires more CPU overhead, and much longer recognition and response times. This is especially true when the CPU is servicing an interrupt that has higher priority than the interrupts from the SIU.

In FLEXIBLE mode, when the SIU receives a frame, it interrupts the CPU. The CPU then reads the control byte from the Receive Control Byte (RCB) register. If the received frame is an information frame, the CPU also reads the information from the receive buffer, according to the values in the Receive Buffer Start (RBS) address register and the Received Field Length (RFL) register.

In FLEXIBLE mode, the 8044 can initiate transmissions without being polled, and thus it can act as the primary station. To initiate transmission or to generate a response, the CPU sets up and enables the SIU. The SIU then formats and transmits the desired frame. Upon completion of the transmission, without waiting for a positive acknowledgement from the receiving station, the SIU interrupts the CPU.

#### 18.5 8044 FRAME FORMAT OPTIONS

As mentioned above, variations on the basic SDLC frame consist of omitting one or more of the fields. The choice of which fields to omit, as well as the selection of AUTO mode versus FLEXIBLE mode, is specified by the settings of the following three bits in the Serial Mode Register (SMD) and the Status/Control Register (STS):

**SMD Bit 0: NFCS (No Frame Check Sequence)**

**SMD Bit 1: NB (Non-Buffered Mode—No Control Field)**

**STS Bit 1: AM (AUTO Mode or Addressed Mode)**

Figure 18-5 shows how these three bits control the frame format.

The following paragraphs discuss some properties of the standard SDLC format, and the significance of omitting some of the fields.

#### 18.5.1 Standard SDLC Format

The standard SDLC format consists of an opening flag, an 8-bit address field, and 8-bit control field, an n-byte information field, a 16-bit Frame Check Sequence (FCS), and a closing flag. The FCS is based on the CCITT-CRC polynomial ( $X^{16} + X^{12} + X^5 + 1$ ). The address and control fields may not be extended. Within the 8044, the address field is held in the Station Address (STAD) register, and the control field is held in the Receive Control Byte (RCB) or Transmit Control Byte (TCB) register. The standard SDLC format may be used in either AUTO mode or FLEXIBLE mode.

#### 18.5.2 No Control Field (Non-Buffered Mode)

When the control field is not present, the RCB and TCB registers are not used. The information field begins immediately after the address field, or, if the address field is also absent, immediately after the opening flag. The entire information field is stored in the 8044's on-chip RAM. If there is no control field, FLEXIBLE mode must be used. Control information may, of course, be present in the information field, and in this manner the No Control Field option may be used for implementing extended control fields.

#### 18.5.3 No Control Field and No Address Field

The No Address Field option is available only in conjunction with the No Control Field option. The STAD, RCB, and TCB registers are not used. When both these fields are absent, the information field begins immediately after the opening flag. The entire information field is stored in on-chip RAM. FLEXIBLE mode must be used. Formats without an address field have the following applications:

Point-to-point data links (where no addressing is necessary)

Monitoring line activity (receiving all messages regardless of the address field)

Extended addressing



FRAME OPTION	NFCS	NB	AM	FRAME FORMAT
Standard SDLC FLEXIBLE Mode	0	0	0	F A C I FCS F
Standard SDLC AUTO Mode	0	0	1	F A C I FCS F
No Control Field FLEXIBLE Mode	0	1	1	F A I FCS F
No Control Field No Address Field FLEXIBLE Mode	0	1	0	F I FCS F
No FCS Field FLEXIBLE Mode	1	0	0	F A C I F
No FCS Field AUTO Mode	1	0	1	F A C I F
No FCS Field No Control Field FLEXIBLE Mode	1	1	1	F A I F
No FCS Field No Control Field No Address Field FLEXIBLE Mode	1	1	0	F I F

**Key to Abbreviations:**

F = Flag (01111110)      I = Information Field  
A = Address Field      FCS = Frame Check Sequence  
C = Control Field

Note: The AM bit is AUTO mode control bit when NB = 0, and Address Mode control bit when NB = 1.

Figure 18-5. Frame Format Options

### 18.5.4 No FCS Field

In the normal case (NFCS=0), the last 16 bits before the closing flag are the Frame Check Sequence (FCS) field. These bits are not stored in the 8044's RAM. Rather, they are used to compute a cyclic redundancy check (CRC) on the data in the rest of the frame. A received frame with a CRC error (incorrect FCS) is ignored. In transmission, the FCS field is automatically computed by the SIU, and placed in the transmitted frame just prior to the closing flag.

The NFCS bit (SMD Bit 0) gives the user the capability of overriding this automatic feature. When this bit is set (NFCS=1), all bits from the beginning of the information field to the beginning of the closing flag are treated as part of the information field, and are stored in the on-chip RAM. No FCS checking is done on the received frames, and no FCS is generated for the transmitted frames. The No FCS Field option may be used in conjunction with any of the other options. It is typically used in FLEXIBLE mode, although it does not strictly include AUTO mode. Use of the No FCS Field option

AUTO Mode may, however, result in SDLC protocol violations, since the data integrity is not checked by the SIU.

Formats without an FCS field have the following applications:

Receiving and transmitting frames without verifying data integrity

Using an alternate data verification algorithm

Using an alternate CRC-16 polynomial (such as  $X^{16} + X^{15} + X^2 + 1$ ), or a 32-bit CRC

Performing data link diagnosis by forcing false CRCs to test error detection mechanisms

In addition to the applications mentioned above, all of the format variations are useful in the support of non-standard bit-synchronous protocols.

## 18.6 HDLC

In addition to its support of SDLC communications, the 8044 also supports some of the capabilities of HDLC. The following remarks indicate the principal differences between SDLC and HDLC.

HDLC permits any number of bits in the information field, whereas SDLC requires a byte structure (multiple of 8 bits). The 8044 itself operates on byte boundaries, and thus it restricts fields to multiples of 8 bits.

HDLC provides functional extensions to SDLC: an unlimited address field is allowed, and extended frame number sequencing.

HDLC does not support operation in loop configurations.

## 18.7 SIU SPECIAL FUNCTION REGISTERS

The 8044 CPU communicates with and controls the SIU through hardware registers. These registers are accessed using direct addressing. The SIU special function registers (SIU SFRs) are of three types:

Control and Status Registers

Parameter Registers

ICE Support Registers

### 18.7.1 Control and Status Registers

There are three SIU Control and Status Registers:

Serial Mode Register (SMD)

Status/Command Register (STS)

### Send/Receive Count Register (NSNR)

The SMD, STS, and NSNR registers are all cleared by system reset. This assures that the SIU will power up in an idle state (neither receiving nor transmitting).

These registers and their bit assignments are described below (see also the More Details on Registers section).

### SMD: Serial Mode Register (byte-addressable)

Bit:	7	6	5	4	3	2	1	0
	SCM2	SCM1	SCM0	NRZI	LOOP	PFS	NB	NFCS

The Serial Mode Register (Address C9H) selects the operational modes of the SIU. The 8044 CPU can both read and write SMD. The SIU can read SMD but cannot write to it. To prevent conflict between CPU and SIU access to SMD, the CPU should write SMD only when the Request To Send (RTS) and Receive Buffer Empty (RBE) bits (in the STS register) are both false (0). Normally, SMD is accessed only during initialization.

The individual bits of the Serial Mode Register are as follows:

Bit #	Name	Description
SMD.0	NFCS	No FCS field in the SDLC frame.
SMD.1	NB	Non-Buffered mode. No control field in the SDLC frame.
SMD.2	PFS	Pre-Frame Sync mode. In this mode, the 8044 transmits two bytes before the first flag of a frame, for DPLL synchronization. If NRZI is enabled, 00H is sent; otherwise, 55H is sent. In either case, 16 pre-frame transitions are guaranteed.
SMD.3	LOOP	Loop configuration.
SMD.4	NRZI	NRZI coding option.
SMD.5	SCM0	Select Clock Mode — Bit 0
SMD.6	SCM1	Select Clock Mode — Bit 1
SMD.7	SCM2	Select Clock Mode — Bit 2

The SCM bits decode as follows:

SCM			Clock Mode	Data Rate (Bits/sec)*
2	1	0		
0	0	0	Externally clocked	0–2.4M**
0	0	1	Undefined	
0	1	0	Self clocked, timer overflow	244–62.5K
0	1	1	Undefined	
1	0	0	Self clocked, external 16x	0–375K

SCM			Clock Mode	Data Rate (Bits/sec)*
2	1	0		
1	0	1	Self clocked, external 32x	0-187.5K
1	1	0	Self clocked, internal fixed	375K
1	1	1	Self clocked, internal fixed	187.5k

\*Based on a 12 Mhz crystal frequency

\*\*0-1M bps in loop configuration

#### STS: Status/Command Register (bit-addressable)

Bit:	7	6	5	4	3	2	1	0
	TBF	RBE	RTS	SI	BOV	OPB	AM	RBP

The Status/Command Register (Address C8H) provides operational control of the SIU by the 8044 CPU, and enables the SIU to post status information for the CPU's access. The SIU can read STS, and can alter certain bits, as indicated below. The CPU can both read and write STS asynchronously. However, 2-cycle instructions that access STS during both cycles ('JBC/B, REL' and 'MOV /B,C') should not be used, since the SIU may write to STS between the two CPU accesses.

The individual bits of the Status/Command Register are as follows:

Bit #	Name	Description
STS.0	RBP	Receive Buffer Protect. Inhibits writing of data into the receive buffer. In AUTO mode, RBP forces an RNR response instead of an RR.
STS.1	AM	AUTO Mode/Addressed Mode. Selects AUTO mode where AUTO mode is allowed. If NB is true, (=1), the AM bit selects the addressed mode. AM may be cleared by the SIU.
STS.2	OPB	Optional Poll Bit. Determines whether the SIU will generate an AUTO response to an optional poll (UP with P=0). OPB may be set or cleared by the SIU.
STS.3	BOV	Receive Buffer Overrun. BOV may be set or cleared by the SIU.
STS.4	SI	SIU Interrupt. This is one of the five interrupt sources to the CPU. The vector location = 23H. SI may be set by the SIU. It should be cleared by the CPU before returning from an interrupt routine.
STS.5	RTS	Request To Send. Indicates that the 8044 is ready to transmit or is trans-

mitting. RTS may be read or written by the CPU. RTS may be read by the SIU, and in AUTO mode may be written by the SIU.

STS.6 RBE Receive Buffer Empty. RBE can be thought of as Receive Enable. RBE is set to one by the CPU when it is ready to receive a frame, or has just read the buffer, and to zero by the SIU when a frame has been received.

STS.7 TBF Transmit Buffer Full. Written by the CPU to indicate that it has filled the transmit buffer. TBF may be cleared by the SIU.

#### NSNR: Send/Receive Count Register (bit-addressable)

Bit:	7	6	5	4	3	2	1	0
	NS2	NS1	NS0	SES	NR2	NR1	NR0	SER

The Send/Receive Count Register (Address D8H) contains the transmit and receive sequence numbers, plus tally error indications. The SIU can both read and write NSNR. The 8044 CPU can both read and write NSNR asynchronously. However, 2-cycle instructions that access NSNR during both cycles ('JBC /B, REL'; and 'MOV /B,C') should not be used, since the SIU may write to NSNR between the two 8044 CPU accesses.

The individual bits of the Send/Receive Count Register are as follows:

Bit #	Name	Description
NSNR.0	SER	Receive Sequence Error: NS (P) ≠ NR (S)
NSNR.1	NR0	Receive Sequence Counter—Bit 0
NSNR.2	NR1	Receive Sequence Counter—Bit 1
NSNR.3	NR2	Receive Sequence Counter—Bit 2
NSNR.4	SES	Send Sequence Error: NR (P) ≠ NS (S) and NR (P) ≠ NS (S) + 1
NSNR.5	NS0	Send Sequence Counter — Bit 0
NSNR.6	NS1	Send Sequence Counter — Bit 1
NSNR.7	NS2	Send Sequence Counter — Bit 2

#### 18.72 Parameter Registers

There are eight parameter registers that are used in connection with SIU operation. All eight registers may be read or written by the 8044 CPU. RFL and RCB are normally loaded by the SIU.

The eight parameter registers are as follows:

#### STAD: Station Address Register (byte-addressable)

The Station Address register (Address CEH) contains the station address. To prevent access conflict, the CPU should access STAD only when the SIU is idle (RTS=0 and RBE=0). Normally, STAD is accessed only during initialization.

#### TBS: Transmit Buffer Start Address Register (byte-addressable)

The Transmit Buffer Start address register (Address DCH) points to the location in on-chip RAM for the beginning of the I-field of the frame to be transmitted. The CPU should access TBS only when the SIU is not transmitting a frame (when TBF=0).

#### TBL: Transmit Buffer Length Register (byte-addressable)

The Transmit Buffer Length register (Address DBH) contains the length (in bytes) of the I-field to be transmitted. A blank I-field (TBL=0) is valid. The CPU should access TBL only when the SIU is not transmitting a frame (when TBF=0).

NOTE: The transmit and receive buffers are not allowed to "wrap around" in the on-chip RAM. A "buffer end" is automatically generated if address 191 (BFH) is reached.

#### TCB: Transmit Control Byte Register (byte-addressable)

The Transmit Control Byte register (Address DAH) contains the byte which is to be placed in the control field of the transmitted frame, during NON-AUTO mode transmission. The CPU should access TCB only when the SIU is not transmitting a frame (when TBF=0). The  $N_S$  and  $N_R$  counters are not used in the NON-AUTO mode.

#### RBS: Receive Buffer Start Address Register (byte-addressable)

The Receive Buffer Start address register (Address CCH) points to the location in on-chip RAM where the beginning of the I-field of the frame being received is to be stored. The CPU should write RBS only when the SIU is not receiving a frame (when RBE=0).

#### RBL: Receive Buffer Length Register (byte-addressable)

The Receive Buffer Length register (Address CBH) contains the length (in bytes) of the area in on-chip

RAM allocated for the received I-field. RBL=0 is valid. The CPU should write RBL only when RBE=0.

#### RFL: Receive Field Length Register (byte-addressable)

The Received Field Length register (Address CDH) contains the length (in bytes) of the received I-field that has just been loaded into on-chip RAM. RFL is loaded by the SIU. RFL=0 is valid. RFL should be accessed by the CPU only when RBE=0.

#### RCB: Receive Control Byte Register (byte-addressable)

The Received Control Byte register (Address CAH) contains the control field of the frame that has just been received. RCB is loaded by the SIU. The CPU can only read RCB, and should only access RCB when RBE=0.

### 18.7.3 ICE Support Registers

The 8044 In-Circuit Emulator (ICE-44) allows the user to exercise the 8044 application system and monitor the execution of instructions in real time.

The emulator operates with Intel's Intellec® development system. The development system interfaces with the user's 8044 system through an in-cable buffer box. The cable terminates in a 8044 pin-compatible plug, which fits into the 8044 socket in the user's system. With the emulator plug in place, the user can exercise his system in real time while collecting up to 255 instruction cycles of real-time data. In addition, he can single-step the program.

Static RAM is available (in the in-cable buffer box) to emulate the 8044 internal and external program memory and external data memory. The designer can display and alter the contents of the replacement memory in the buffer box, the internal data memory, and the internal 8044 registers, including the SFRs.

Among the SIU SFRs are the following registers that support the operation of the ICE:

#### DMA CNT: DMA Count Register (byte-addressable)

The DMA Count register (Address CFH) indicates the number of bytes remaining in the information block that is currently being used.

#### FIFO: Three-Byte (byte-addressable)

The Three-Byte FIFO (Address DDH, DEH, and DFH) is used between the eight-bit shift register and the information buffer when an information block is received.



# SIUST: SIU State Counter (byte-addressable)

The SIU State Counter (Address D9H) reflects the state of the internal logic which is under SIU control. Therefore, care must be taken not to write into this register.

The SIUST register can serve as a helpful aid to determine which field of a receive frame that the SIU expects next. The table below will help in debugging 8044 reception problems.

## SIUST

### VALUE FUNCTION

01H	Waiting for opening flag.
08H	Waiting for address field.
10H	Waiting for control field.
18H	Waiting for first byte of I field. This state is only entered if a FCS is expected. It pushes the received byte onto the top of the FIFO.
20H	Waiting for second byte of I field. This state always follows state 18H.
28H	Waiting for I field byte. This state can be en-

tered from state 20H or from states 01H, 08H, or 10H depending upon the SIU's mode configuration. (Each time a byte is received, it is pushed onto the top of the FIFO and the byte at the bottom is put into memory. For no FCS formatted frames, the FIFO is collapsed into a single register).

30H Waiting for the closing flag after having overflowed the receive buffer. Note that even if the receive frame overflows the assigned receive buffer length, the FCS is still checked.

Examples of SIUST status sequences for different frame formats are shown below. Note that status changes after acceptance of the received field byte.

## 18.8 OPERATION

The SIU is initialized by a reset signal (on pin 9), followed by write operations to the SIU SFRs. Once initialized, the SIU can function in AUTO mode or NON-AUTO mode. Details are given below.

Table 18-1. SIUST Status Sequences

Example 1:

Frame Format

SIUST Value

(Idle)	F	A	C	I			FCS	F
01	01	08	10	18	20	28	28	01

Example 2:

Frame Format

SIUST Value

(Idle)	F	A	I			FCS	F
01	01	08	18	20	28	28	01

Example 3:

Frame Format

SIUST Value

(Idle)	F		I			FCS	F
01	01	18	20	28	28	01	

Example 4:

Frame Format

SIUST Value

(Idle)	F	A	I	F				
01	01	08	28	01				

Example 5:

Frame Format

SIUST Value

(Idle)	F		I	F				
01	01		28	01				

Example 6:

Frame Format

SIUST Value

(Idle)	F	I		I OVERFLOW	FCS	F
01	01	18	20	28	30	01

### Frame Option

NFCS NB AM

0 0 1

0 1 1

0 1 1

0 1 1

0 1 0

0 1 0

0 1 0

0 1 0

### 18.8.1 Initialization

Figure 18-6 is the SIU. Registers SMD, STS, and NSNR are cleared by reset. This puts the 8044 into an idle state—neither receiving nor transmitting. The following registers must be initialized before the 8044 leaves the idle state:

**STAD**—to establish the 8044's SDLC station address.

**SMD**—to configure the 8044 for the proper operating mode.

**RBS, RBL**—to define the area in RAM allocated for the Receive Buffer.

**TBS, TBL**—to define the area in RAM allocated for the Transmit Buffer.

Once these registers have been initialized, the user may write to the STS register to enable the SIU to leave the idle state, and to begin transmits and/or receives.

Setting RBE to 1 enables the SIU for receive. When RBE = 1, the SIU monitors the received data stream for a flag pattern. When a flag pattern is found, the SIU enters Receive mode and receives the frame.

Setting RTS to 1 enables the SIU for transmit. When RTS = 1, the SIU monitors the received data stream for a GA pattern (loop configuration) or waits for a CTS

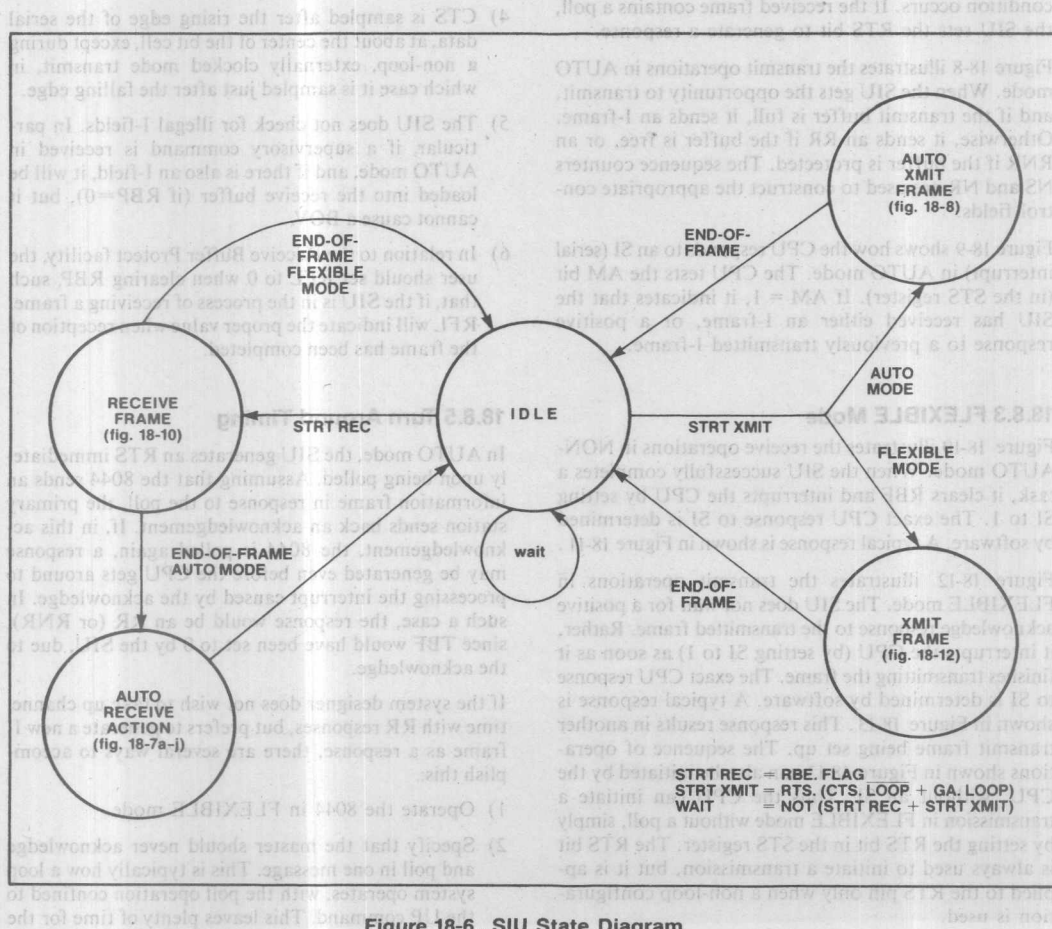


Figure 18-6. SIU State Diagram

(non-loop configuration). When the GA or CTS arrives, the SIU enters Transmit mode and transmits a frame.

In AUTO mode, the SIU sets RTS to enable automatic transmissions of appropriate responses.

### 18.8.2 AUTO Mode

Figure 18-7 illustrates the receive operations in AUTO mode. The overall operation is shown in Figure 18-7a. Particular cases are illustrated in Figures 18-7b through 18-7j. If any Unnumbered Command other than UP is received, the AM bit is cleared and the SIU responds as if in the FLEXIBLE mode, by interrupting the CPU for supervision. This will also happen if a BOV or SES condition occurs. If the received frame contains a poll, the SIU sets the RTS bit to generate a response.

Figure 18-8 illustrates the transmit operations in AUTO mode. When the SIU gets the opportunity to transmit, and if the transmit buffer is full, it sends an I-frame. Otherwise, it sends an RR if the buffer is free, or an RNR if the buffer is protected. The sequence counters NS and NR are used to construct the appropriate control fields.

Figure 18-9 shows how the CPU responds to an SI (serial interrupt) in AUTO mode. The CPU tests the AM bit (in the STS register). If AM = 1, it indicates that the SIU has received either an I-frame, or a positive response to a previously transmitted I-frame.

### 18.8.3 FLEXIBLE Mode

Figure 18-10 illustrates the receive operations in NON-AUTO mode. When the SIU successfully completes a task, it clears RBF and interrupts the CPU by setting SI to 1. The exact CPU response to SI is determined by software. A typical response is shown in Figure 18-11.

Figure 18-12 illustrates the transmit operations in FLEXIBLE mode. The SIU does not wait for a positive acknowledge response to the transmitted frame. Rather, it interrupts the CPU (by setting SI to 1) as soon as it finishes transmitting the frame. The exact CPU response to SI is determined by software. A typical response is shown in Figure 18-13. This response results in another transmit frame being set up. The sequence of operations shown in Figure 18-13 can also be initiated by the CPU, without an SI. Thus the CPU can initiate a transmission in FLEXIBLE mode without a poll, simply by setting the RTS bit in the STS register. The RTS bit is always used to initiate a transmission, but it is applied to the RTS pin only when a non-loop configuration is used.

### 18.8.4 8044 Data Link Particulars

The following facts should be noted:

- 1) In a non-loop configuration, one or two bits are transmitted before the opening flag. This is necessary for NRZI synchronization.
- 2) In a non-loop configuration, one to eight extra dribble bits are transmitted after the closing flag. These bits are a zero followed by ones.
- 3) In a loop configuration, when a GA is received and the 8044 begins transmitting, the sequence is 0111111010111110... (FLAG, 1, FLAG, ADDRESS, etc.). The first flag is created from the GA. The second flag begins the message.
- 4) CTS is sampled after the rising edge of the serial data, at about the center of the bit cell, except during a non-loop, externally clocked mode transmit, in which case it is sampled just after the falling edge.
- 5) The SIU does not check for illegal I-fields. In particular, if a supervisory command is received in AUTO mode, and if there is also an I-field, it will be loaded into the receive buffer (if RBP=0), but it cannot cause a BOV.
- 6) In relation to the Receive Buffer Protect facility, the user should set RFL to 0 when clearing RBP, such that, if the SIU is in the process of receiving a frame, RFL will indicate the proper value when reception of the frame has been completed.

### 18.8.5 Turn Around Timing

In AUTO mode, the SIU generates an RTS immediately upon being polled. Assuming that the 8044 sends an information frame in response to the poll, the primary station sends back an acknowledgement. If, in this acknowledgement, the 8044 is polled again, a response may be generated even before the CPU gets around to processing the interrupt caused by the acknowledge. In such a case, the response would be an RR (or RNR), since TBF would have been set to 0 by the SIU, due to the acknowledge.

If the system designer does not wish to take up channel time with RR responses, but prefers to generate a new I-frame as a response, there are several ways to accomplish this:

- 1) Operate the 8044 in FLEXIBLE mode.
- 2) Specify that the master should never acknowledge and poll in one message. This is typically how a loop system operates, with the poll operation confined to the UP command. This leaves plenty of time for the

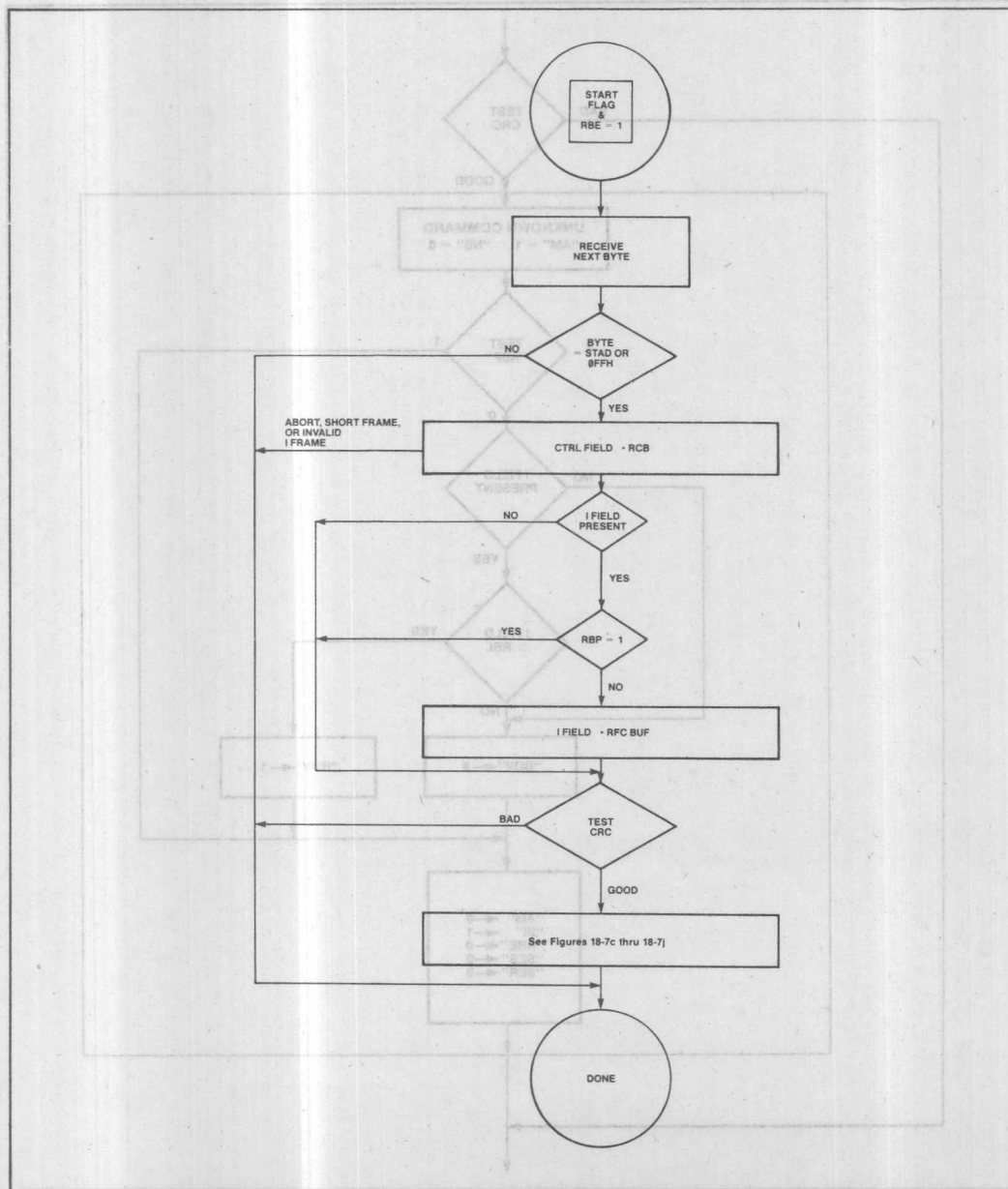


Figure 18-7a. SIU AUTO Mode Receive Flowchart—General



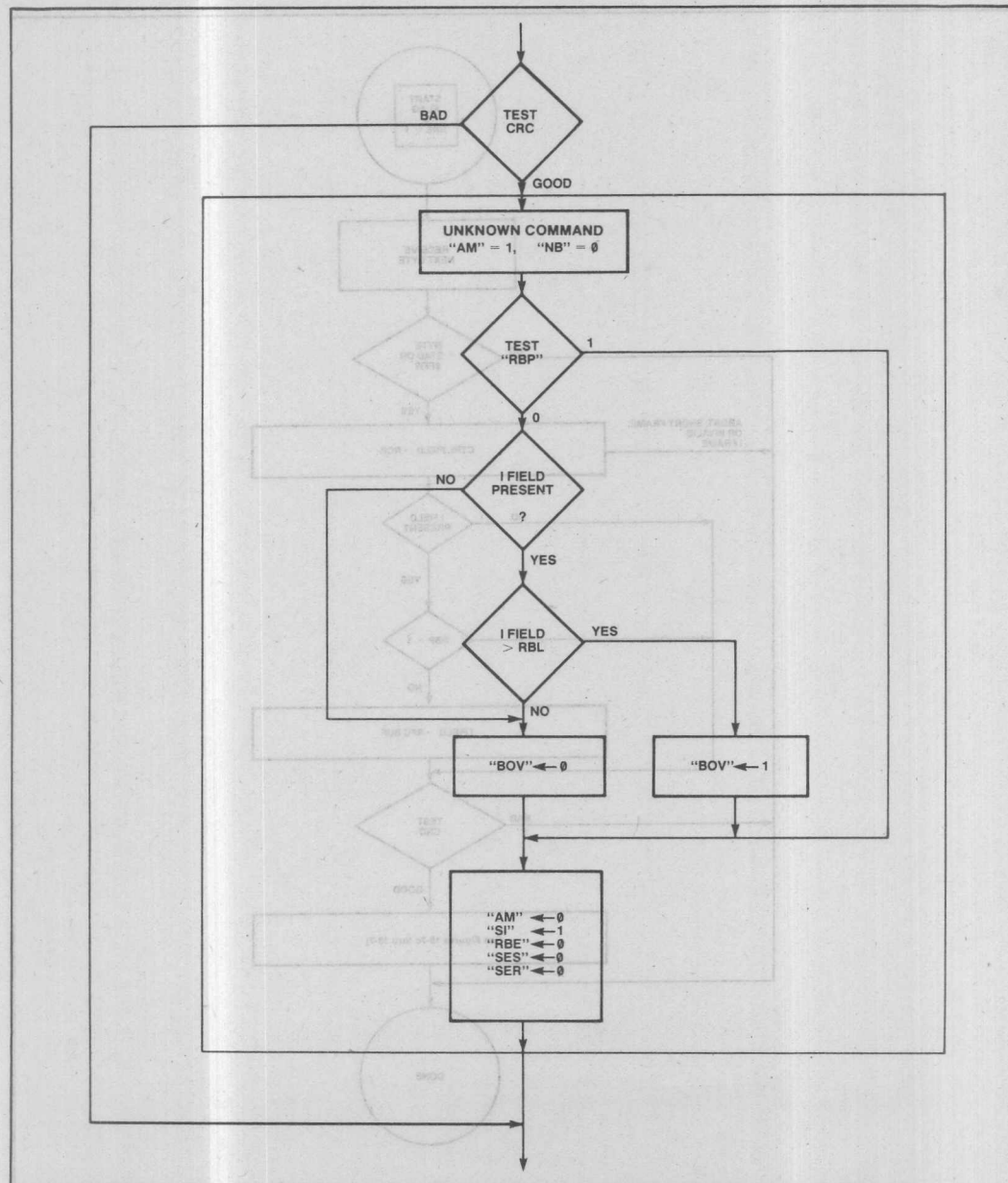


Figure 18-7b. SIU AUTO Mode Receive Flowchart—Unknown Command

18-17

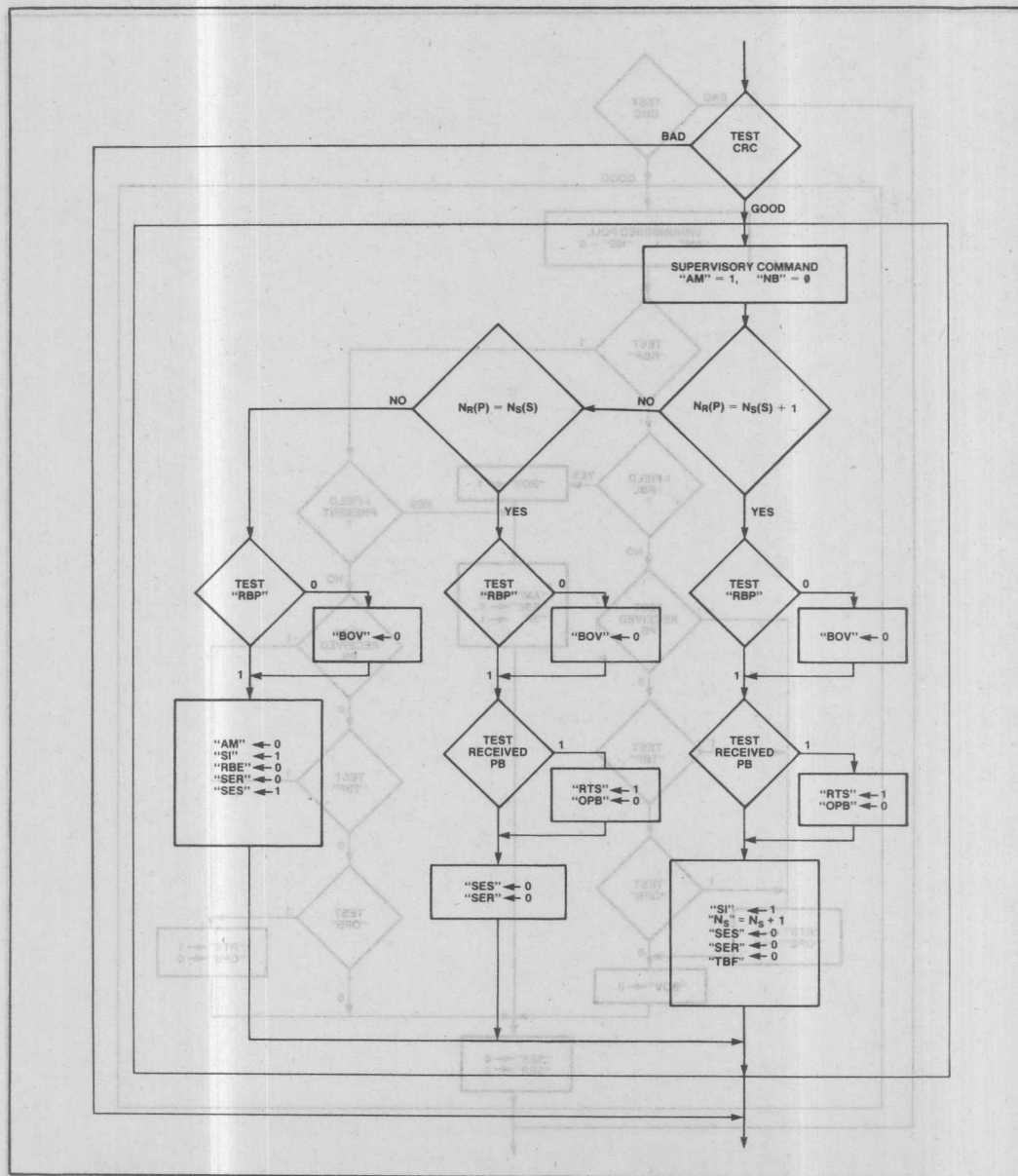


Figure 18-7d. SIU AUTO Mode Receive Flowchart—Supervisory Command

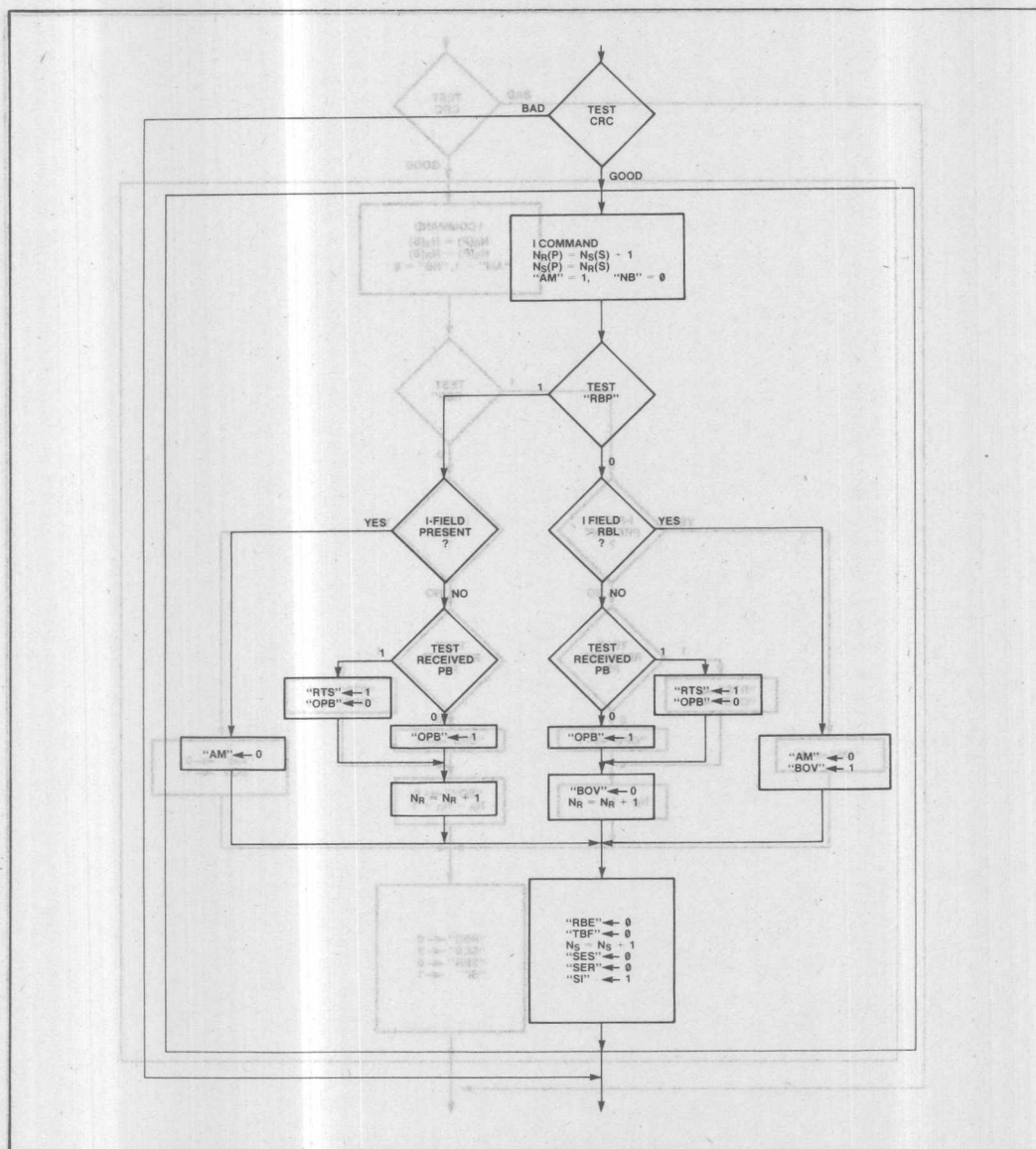


Figure 18-7e. SIU AUTO Mode Receive Flowchart—I Command: Prior Transmitted I-Field Confirmed, Current Received I-Field in Sequence



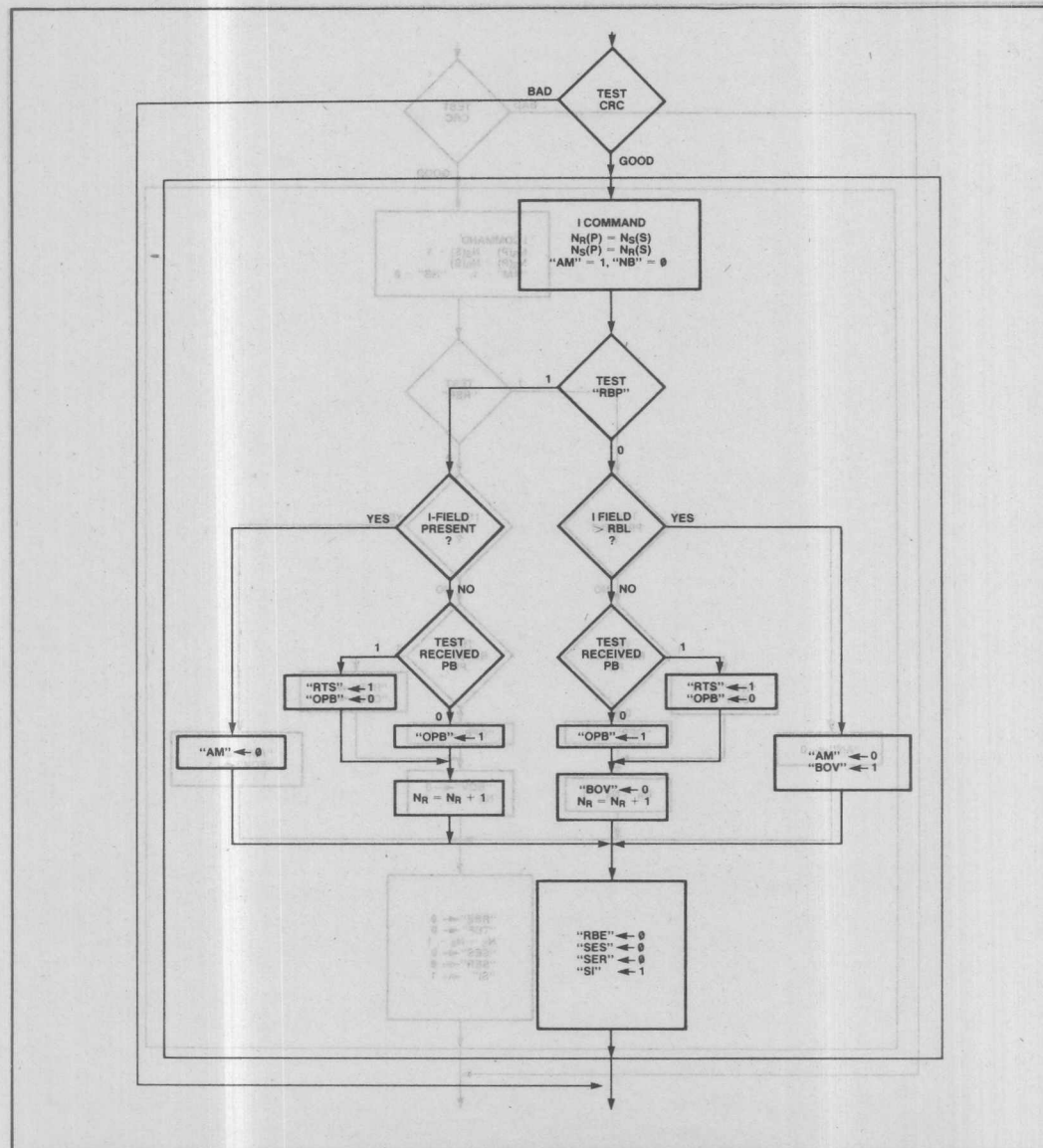


Figure 18-7f. SIU AUTO Mode Receive Flowchart—I Command: Prior Transmitted I-Field Not Confirmed, Current Received I-Field in Sequence

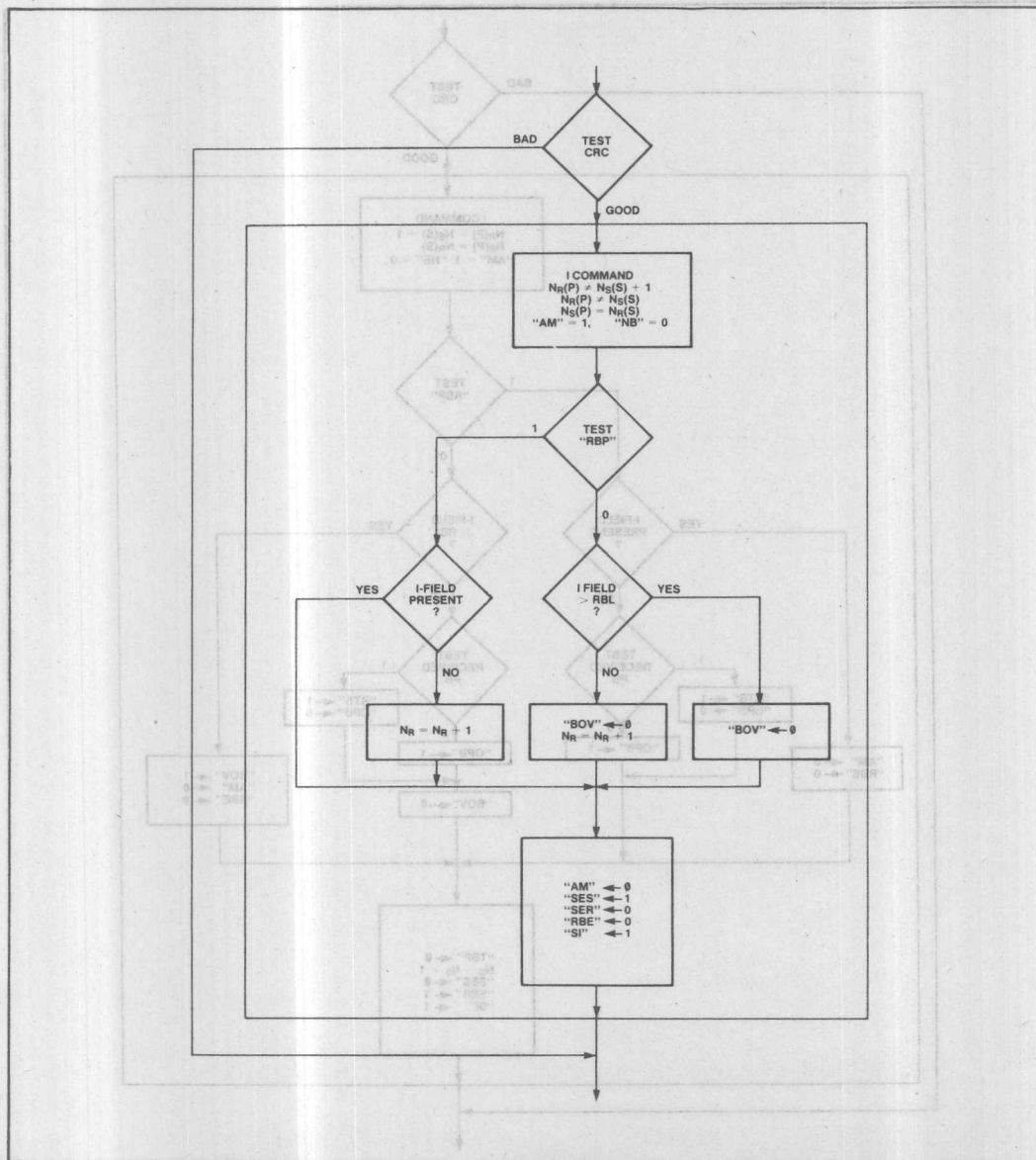


Figure 18-7g. SIU AUTO Mode Receive Flowchart—I Command: Sequence Error Send, Current Received I-Field in Sequence

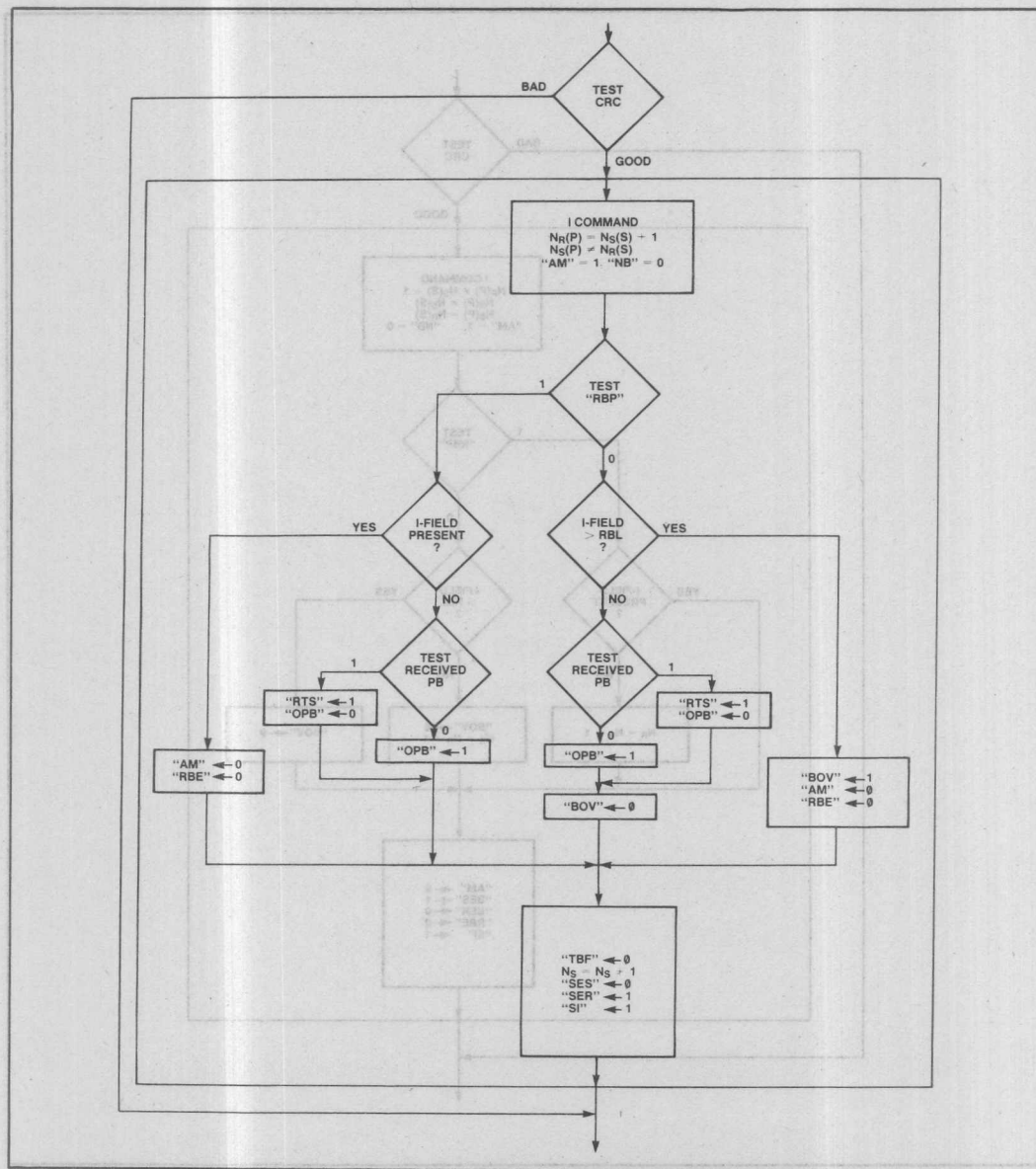
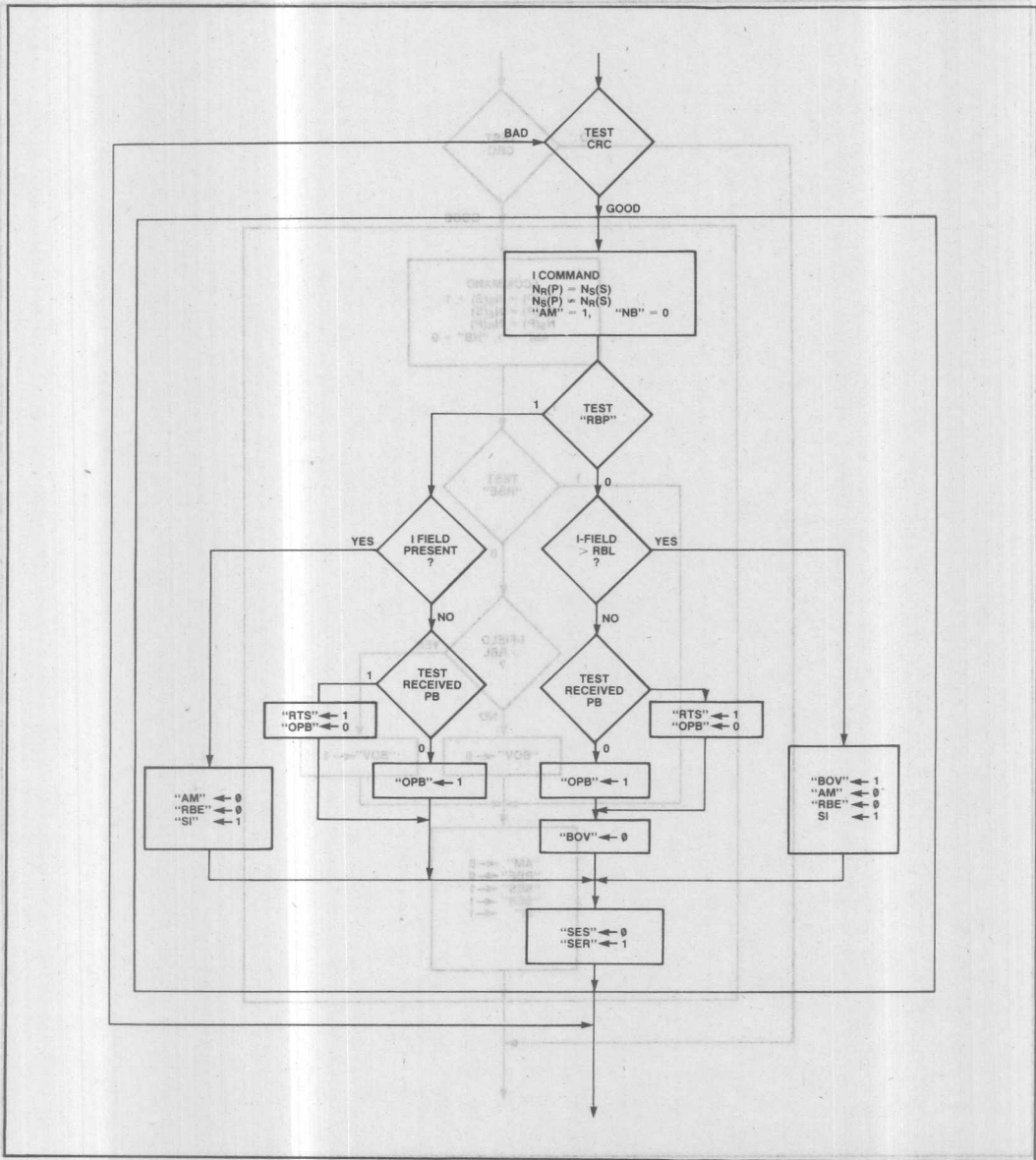


Figure 18-7h. SIU AUTO Mode Receive Flowchart—I Command: Prior Transmitted I-Field Confirmed Sequence Error Receive



**Figure 18-71. SIU AUTO Mode Receive Flowchart—I Command: Prior Transmitted I-Field Not Confirmed, Sequence Error Receive**



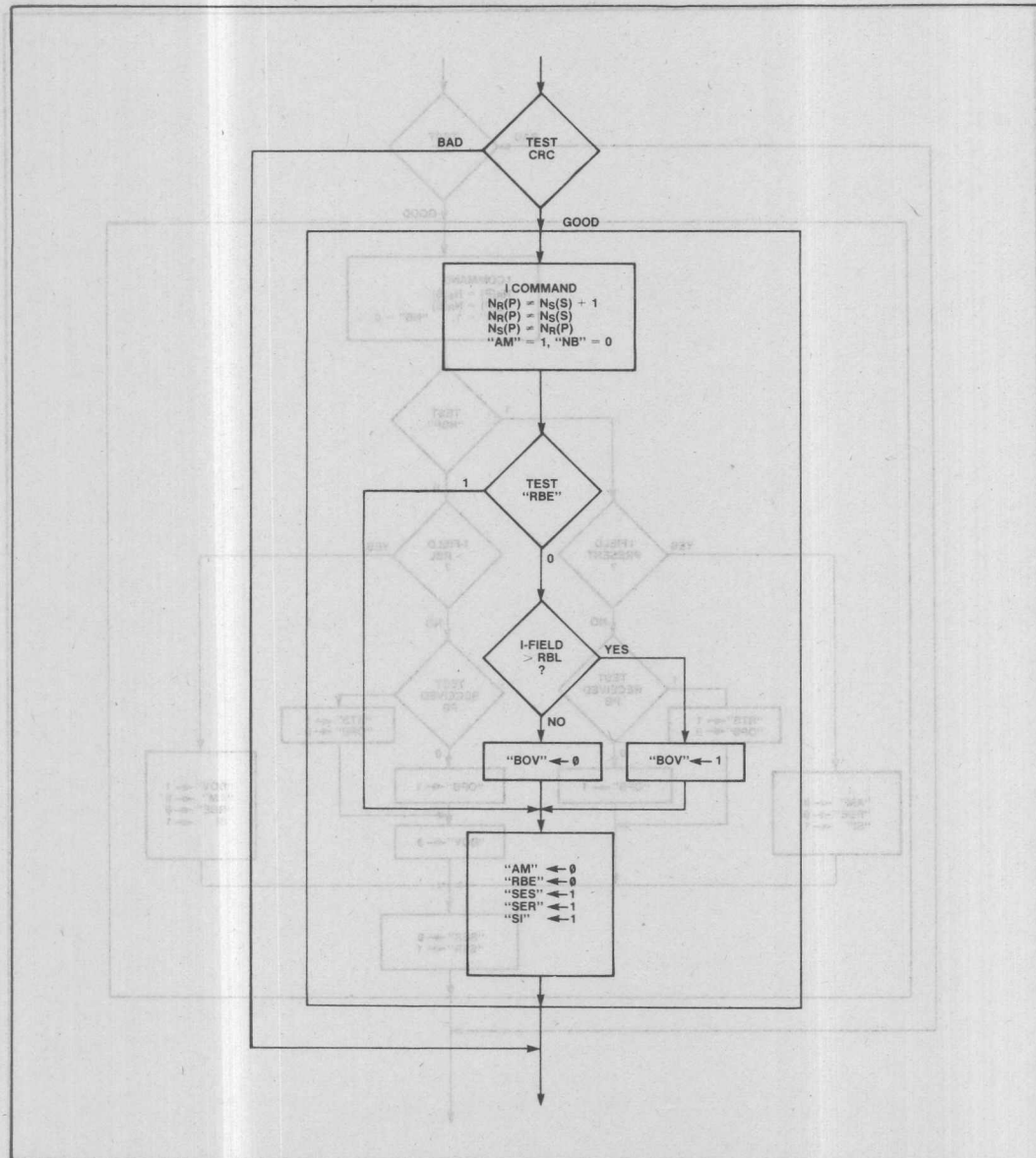


Figure 18-7j. SIU SUTO Mode Receive Flowchart—I Command: Sequence Error Send and Sequence Error Receive

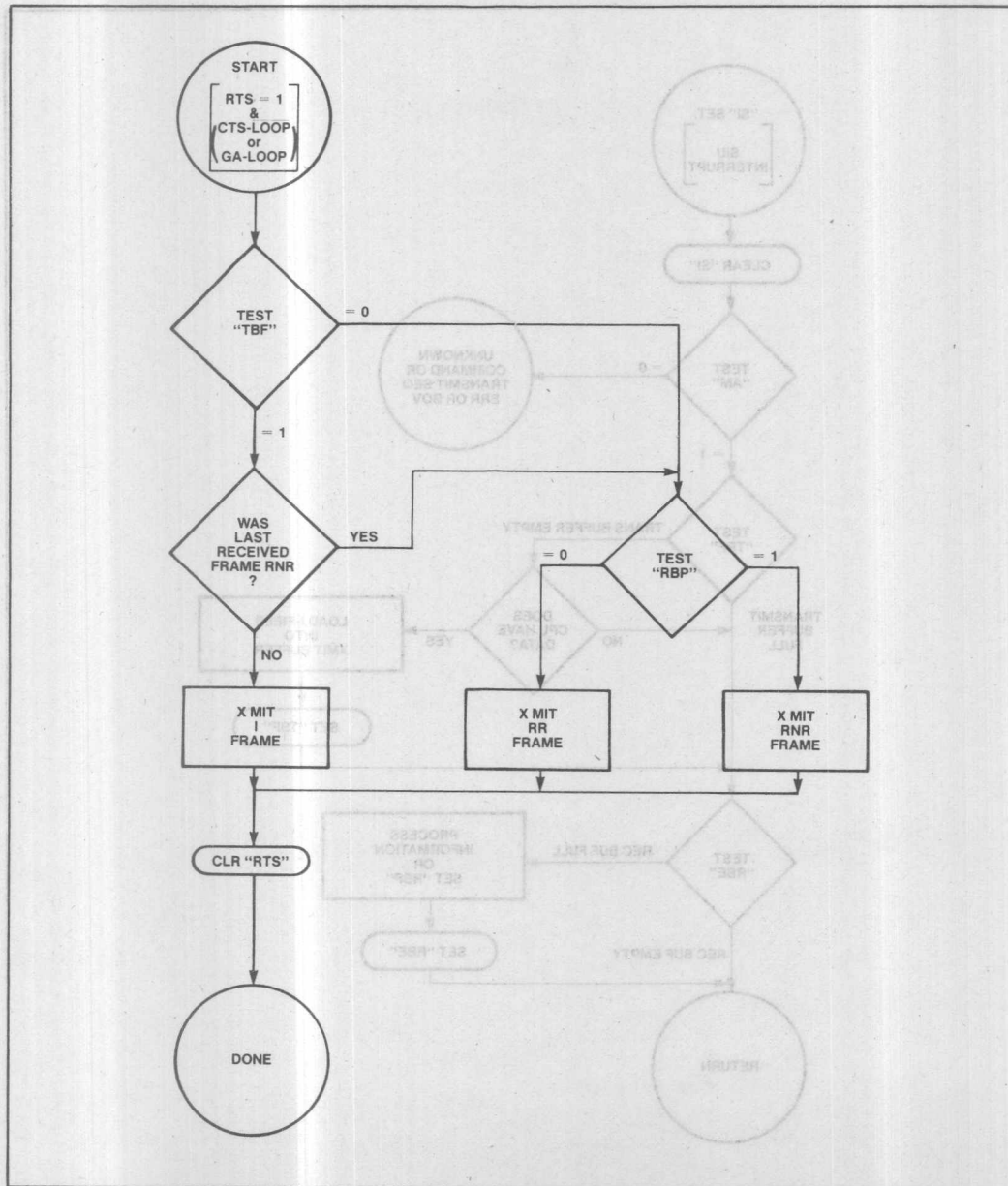


Figure 18-8. SIU AUTO Mode Transmit Flowchart

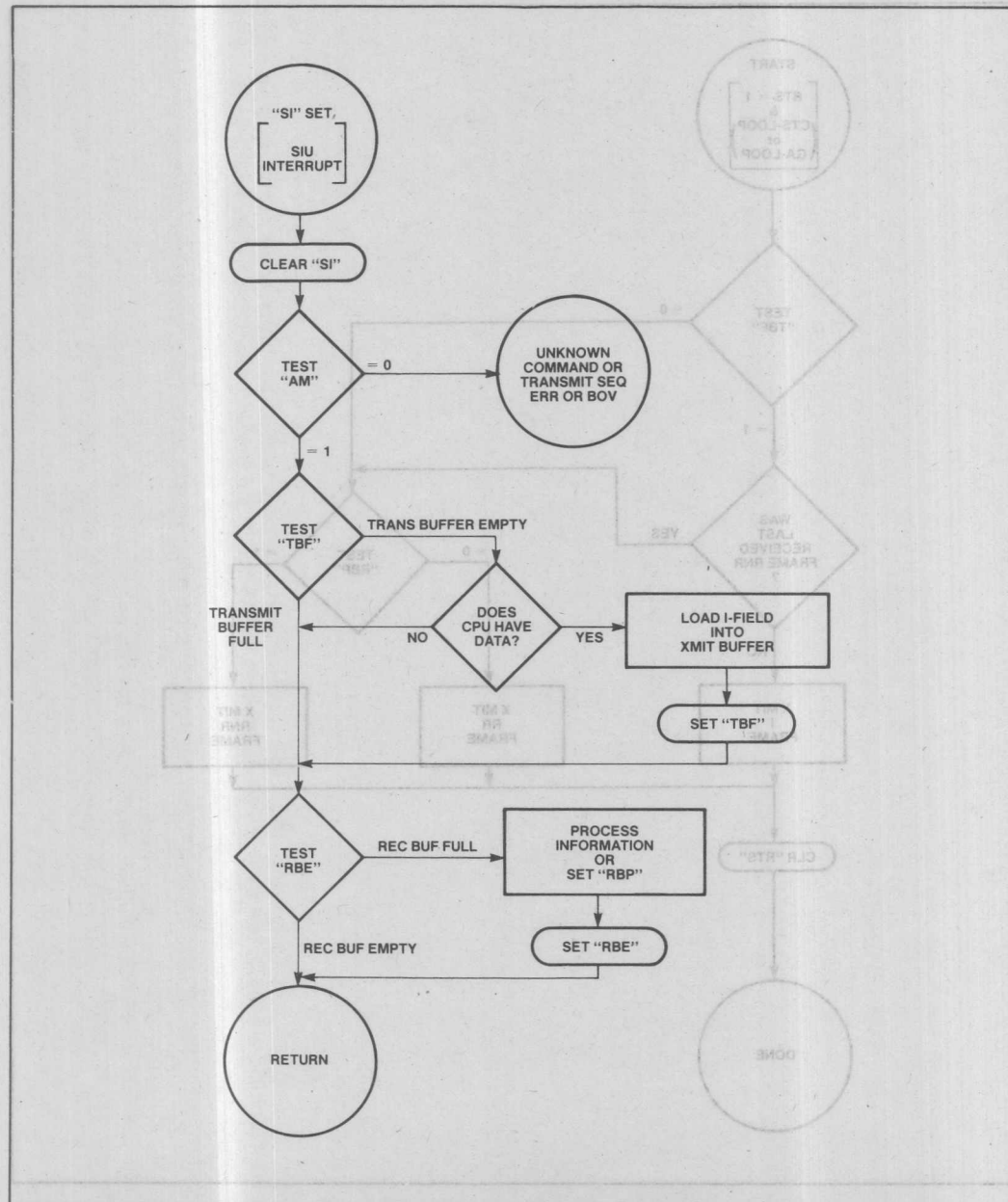


Figure 18-9. AUTO Mode Response to "SI"

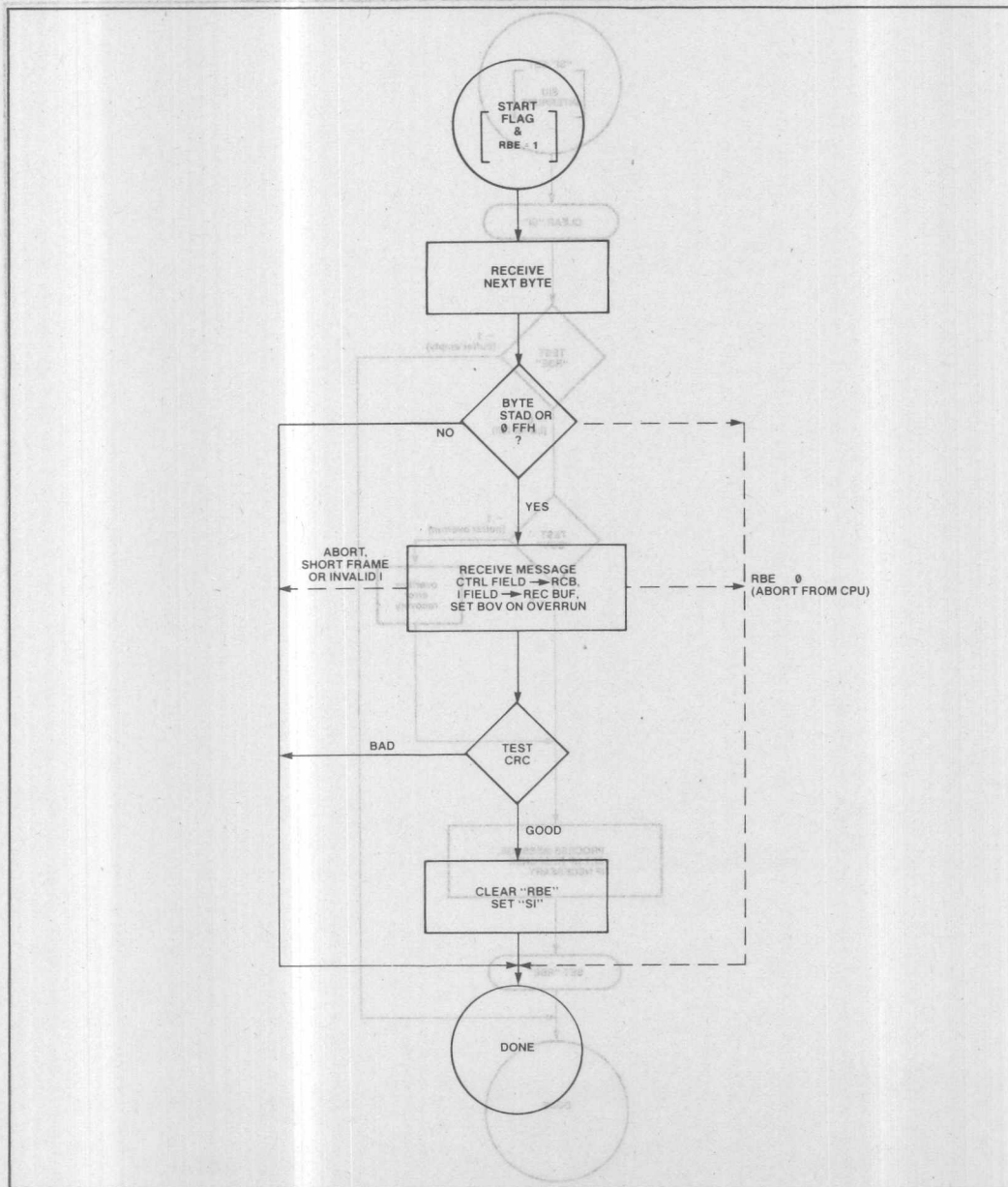


Figure 18-10. SIU FLEXIBLE Mode Receive Flowchart



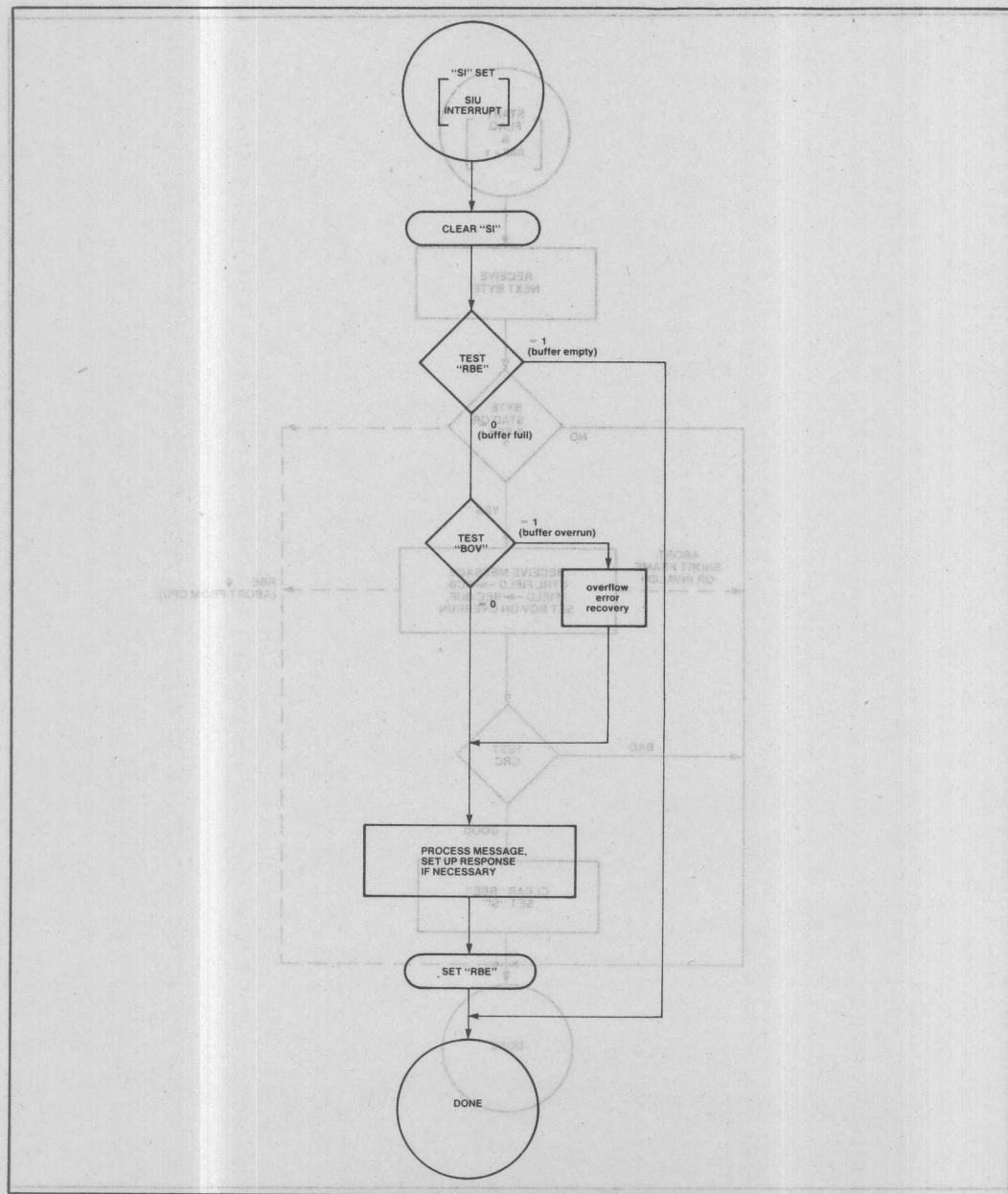


Figure 18-11. FLEXIBLE Mode Response to Receive "SI"

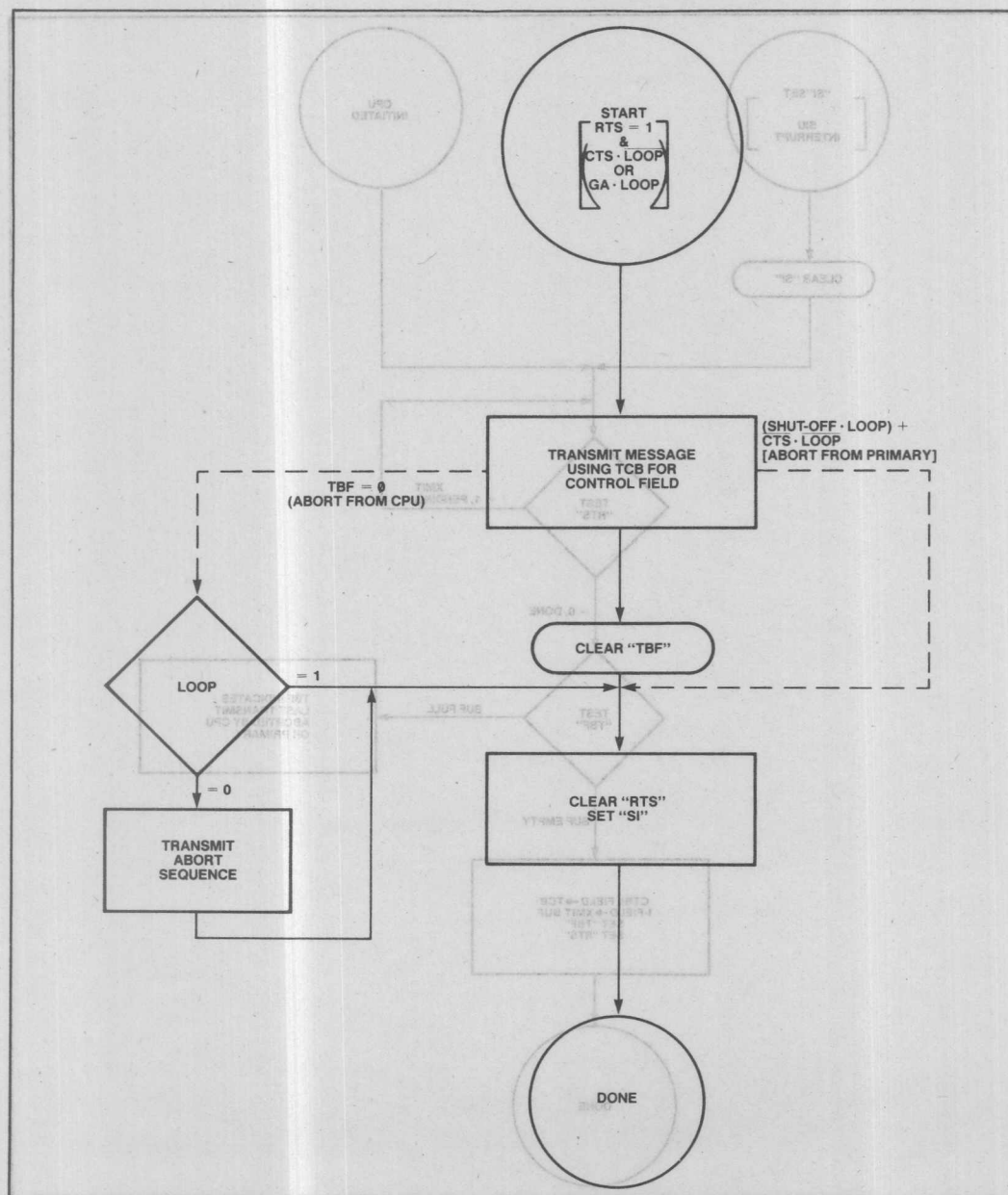


Figure 18-12. SIU FLEXIBLE Mode Transmit Flowchart

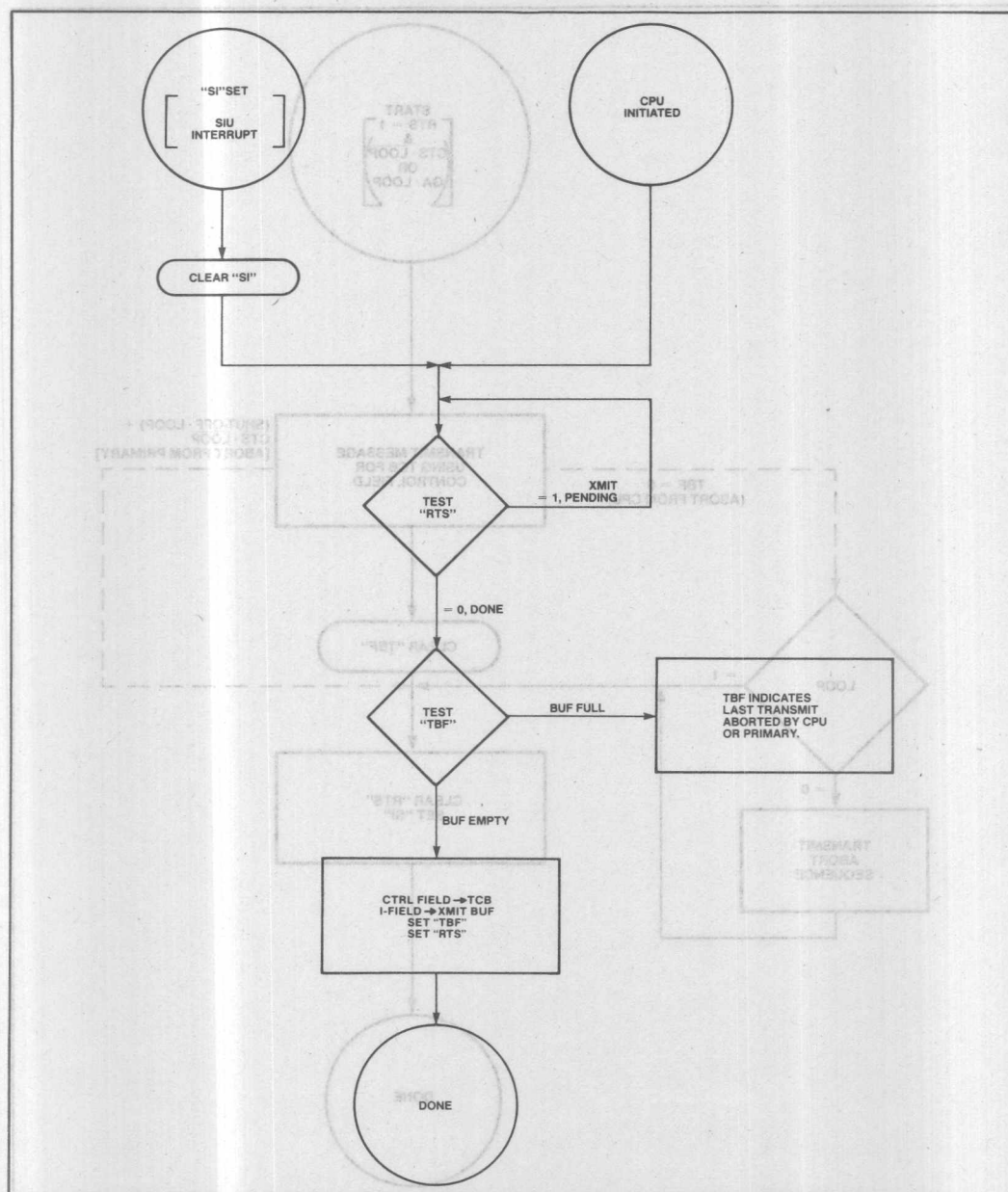


Figure 18-13. FLEXIBLE Mode Response to Transmit "SI"

8044 to get its transmit buffer loaded with new information after an acknowledge.

- 3) The 8044 CPU can clear RTS. This will prevent a response from being sent, or abort it if it is already in progress. A system using external RTS/CTS handshaking could use a one-shot to delay RTS or CTS, thereby giving the CPU more time to disable the response.

## 18.9 MORE DETAILS ON SIU HARDWARE

The SIU divides functionally into two sections—a bit processor (BIP) and a byte processor (BYP)—sharing some common timing and control logic. As shown in Figure 18-14, the BIP operates between the serial port pins and the SIU bus, and performs all functions necessary to transmit/receive a byte of data to/from the serial data stream. These operations include shifting, NRZI encoding/decoding, zero insertion/deletion, and FCS generation/checking. The BYP manipulates bytes of data to perform message formatting, and other transmitting and receiving functions. It operates between the SIU bus (SIB) and the 8044's internal bus (IB). The interface between the SIU and the CPU involves an interrupt and some locations in on-chip RAM space which are managed by the BYP.

The maximum possible data rate for the serial port is limited to 1/2 the internal clock rate. This limit is imposed by both the maximum rate of DMA to the on-chip RAM, and by the requirements of synchronizing to an external clock. The internal clock rate for an 8044 running on a 12 MHz crystal is 6 MHz. Thus the maximum 8044 serial data rate is 3 MHz. This data rate drops down to 2.4 MHz when time is allowed for external clock synchronization.

### 18.9.1 The Bit Processor

In the asynchronous (self clocked) modes the clock is extracted from the data stream using the on-chip digital phase-locked-loop (DPLL). The DPLL requires a clock input at 16 times the data rate. This 16 × clock may originate from SCLK, Timer 1 Overflow, or PH2 (one half the oscillator frequency). The extra divide-by-two described above allows these sources to be treated alternatively as 32 × clocks.

The DPLL is a free-running four-bit counter running off the 16 × clock. When a transition is detected in the receive data stream, a count is dropped (by suppressing the carry-in) if the current count value is greater than 8. A count is added (by injecting a carry into the second stage rather than the first) if the count is less than 8. No

adjustment is made if the transition occurs at the count of 8. In this manner the counter locks in on the point at which transitions in the data stream occur at the count of 8, and a clock pulse is generated when the count overflows to 0.

In order to perform NRZI decoding, the NRZI decoder compares each bit of input data to the previous bit. There are no clock delays in going through the NRZI decoder.

The zero insert/delete circuitry (ZID) performs zero insertion/deletion, and also detects flags, GA's (Go-Ahead's), and aborts (same as GA's) in the data stream. The pattern 1111110 is detected as an early GA, so that the GA may be turned into a flag for loop mode transmission.

The shut-off detector monitors the receive data stream for a sequence of eight zeros, which is a shut-off command for loop mode transmissions. The shut-off detector is a three-bit counter which is cleared whenever a one is found in the receive data stream. Note that the ZID logic could not be used for this purpose, because the receive data must be monitored even when the ZID is being used for transmission.

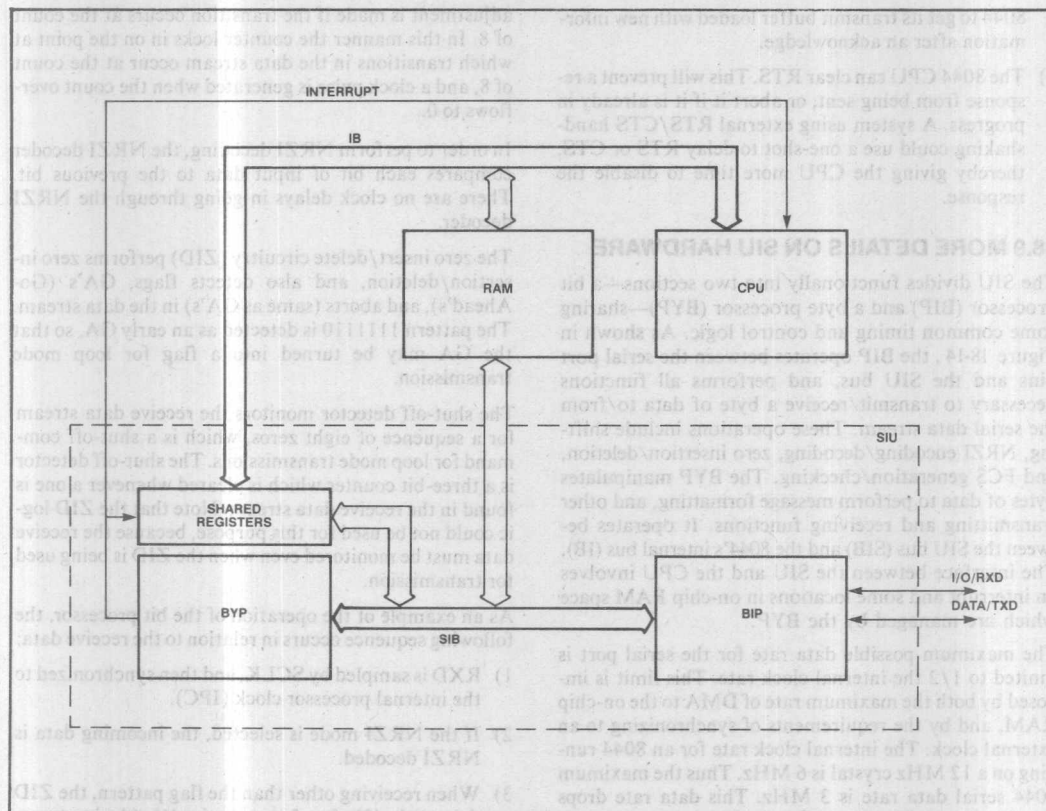
As an example of the operation of the bit processor, the following sequence occurs in relation to the receive data:

- 1) RXD is sampled by SCLK, and then synchronized to the internal processor clock (IPC).
- 2) If the NRZI mode is selected, the incoming data is NRZI decoded.
- 3) When receiving other than the flag pattern, the ZID deletes the '0' after 5 consecutive '1's (during transmission this zero is inserted). The ZID locates the byte boundary for the rest of the circuitry. The ZID deletes the '0's by preventing the SR (shift register) from receiving a clocking pulse.
- 4) The FCS (which is a function of the data between the flags—not including the flags) is initialized and started at the detection of the byte boundary at the end of the opening flag. The FCS is computed each bit boundary until the closing flag is detected. Note that the received FCS has gone through the ZID during transmission.

### 18.9.2 The Byte Processor

Figure 18-15 is a block diagram of the byte processor (BYP). The BYP contains the registers and controllers necessary to perform the data manipulations associated with SDLC communications. The BYP registers may be read or written by the CPU over the 8044's internal bus





**Figure 18-14 . The Bit and Byte Processors**

(IB), using standard 8044 hardware register operations. The 8044 register select PLA controls these operations. Three of the BYP registers connect to the IB through the IBS, a sub-bus which also connects to the CPU interrupt control registers.

Simultaneous access of a register by both the IB and the SIB is prevented by timing. In particular, RAM access is restricted to alternate internal processor cycles for the CPU and the SIU, in such a way that collisions do not occur.

As an example of the operation of the byte processor, the following sequence occurs in relation to the receive data:

- 1) Assuming that there is an address field in the frame, the BYP takes the station address from the register file into temporary storage. After the opening flag,

the next field (the address field) is compared to the station address in the temporary storage. If a match occurs, the operation continues.

- 2) Assuming that there is a control field in the frame, the BYP takes the next byte and loads it into the RCB register. The RCB register has the logic to update the NSNR register (increment receive count, set SES and SER flags, etc.).
- 3) Assuming that there is an information field, the next byte is dumped into RAM at the RBS location. The DMA CNT (RBL at the opening flag) is loaded from the DMA CNT register into the RB register and decremented. The RFL is then loaded into the RB register, incremented, and stored back into the register file.

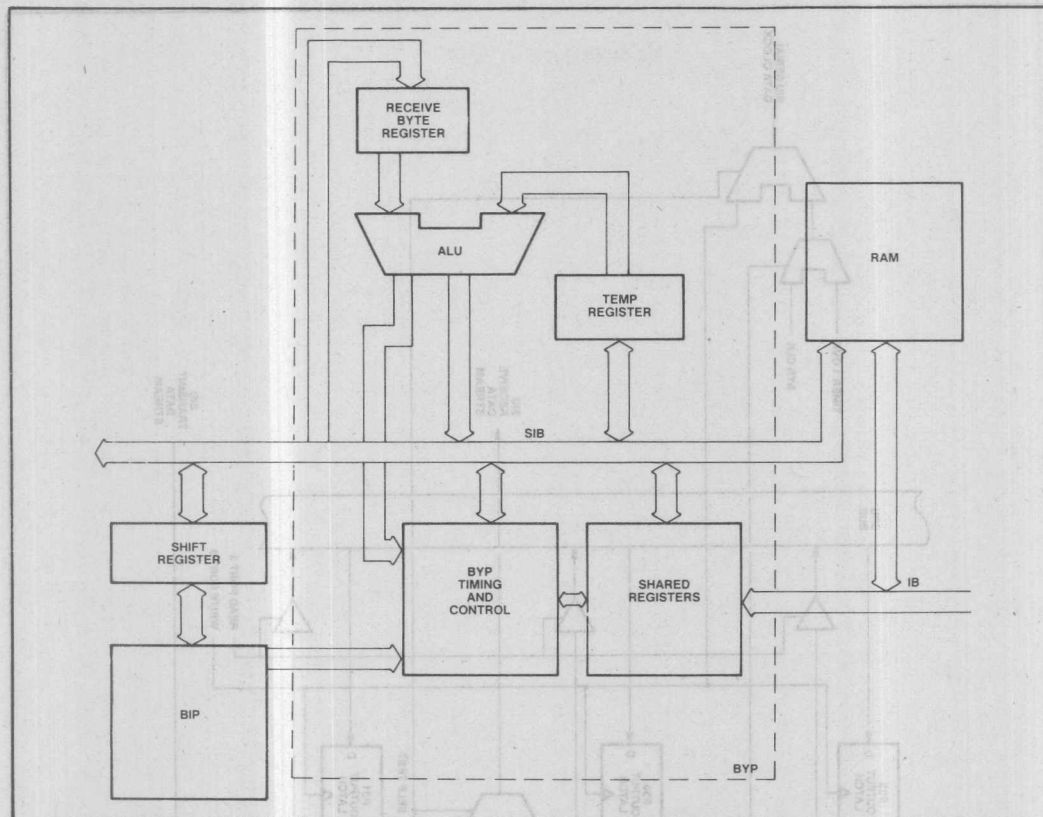


Figure 18-15. The Byte Processor

- 4) This process continues until the DMA CNT reaches zero, or until a closing flag is received. Upon either event, the BYP updates the status, and, if the CRC is good, the NSNR register.

## 18.10 DIAGNOSTICS

An SIU test mode has been provided, so that the on-chip CPU can perform limited diagnostics on the SIU. The test mode utilizes the output latches for P3.0 and P3.1 (pins 10 and 11). These port 3 pins are not useful as out-put ports, since the pins are taken up by the serial port functions. Figure 18-16 shows the signal routing associated with the SIU test mode.

Writing a 0 to P3.1 enables the serial test mode (P3.1 is set to 1 by reset). In test mode the P3.0 bit is mapped

into the received data stream, and the 'write port 3' control signal is mapped into the SCLK path in place of T1. Thus, in test mode, the CPU can send a serial data stream to the SIU by writing to P3.0. The transmit data stream can be monitored by reading P3.1. Each successive bit is transmitted from the SIU by writing to any bit in Port 3, which generates SCLK.

In test mode, the P3.0 and P3.1 pins are placed in a high voltage, high impedance state. When the CPU reads P3.0 and P3.1 the logic level applied to the pin will be returned. In the test mode, when the CPU reads 3.1, the transmit data value will be returned, not the voltage on the pin. The transmit data remains constant for a bit time. Writing to P3.0 will result in the signal being out-putted for a short period of time. However, since the signal is not latched, P3.0 will quickly return to a high voltage, high impedance state.

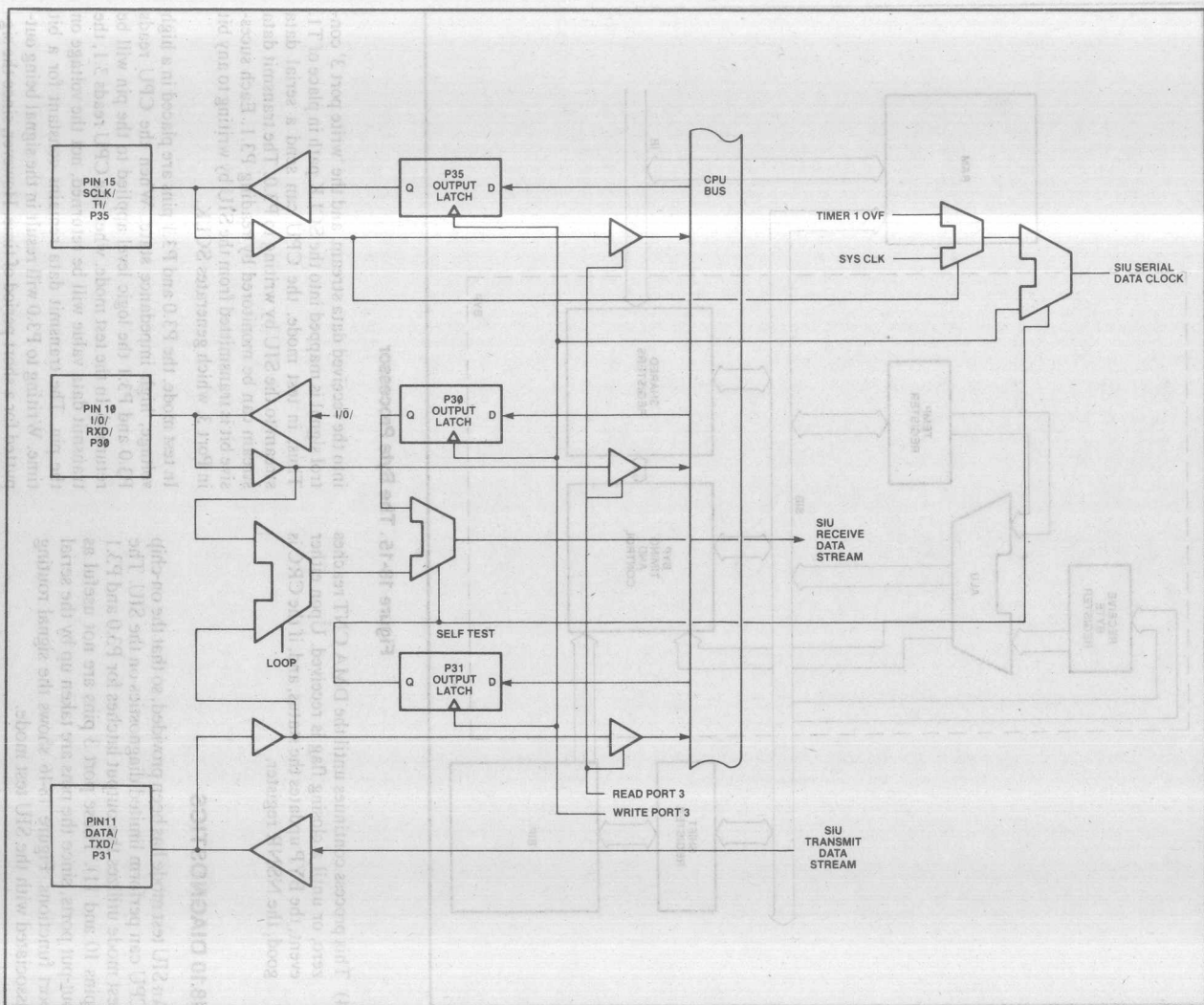


Figure 18-16. SIU Test Mode

The serial test mode is disabled by writing a 1 to P3.1. Care must be taken that a 0 is never written to P3.1 in the course of normal operation, since this causes the test mode to be entered.

Figure 18-17 is an example of a simple program segment that can be imbedded into the user's diagnostic program. That example shows how to put the 8044 into "Loop-back mode" to test the basic transmitting and receiving functions of the SIU.

Loop-back mode is functionally equivalent to a hardware connection between pins 10 and 11 on the 8044.

In this example, the 8044 CPU plays the role of the primary station. The SIU is in the AUTO mode. The CPU sends the SIU a supervisory frame with the poll bit set and an RNR command. The SIU responds with a supervisory frame with the poll bit set and an RR command.

The operation proceeds as follows:

Interrupts are disabled, and the self test mode is enabled by writing a zero to P3.1. This establishes P3.0 as the data path from the CPU to the SIU. CTS (clear-to-send) is enabled by writing a zero to P1.7. The station address is initialized by writing 08AH into the STAD (station address register).

The SIU is configured for receive operation in the clocked mode and in AUTO mode. The CPU then transmits a supervisory frame. This frame consists of an

opening flag, followed by the station address, a control field indicating that this is a supervisory frame with an RNR command, and then a closing flag.

Each byte of the frame is transmitted by writing that byte into the A register and then calling the subroutine XMIT8. Two additional SCLKs are generated to guarantee that the last bits in the frame have been clocked into the SIU. Finally the CPU reads the status register (STS). If the operation has proceeded correctly, the status will be 072H. If it is not, the program jumps to the ERROR loop and terminates.

The SIU generates an SI (SIU interrupt) to indicate that it has received a frame. The CPU clears this interrupt, and then begins to monitor the data stream that is being generated by the SIU in response to what it has received. As each bit arrives (via P3.1), it is moved into the accumulator, and the CPU compares the byte in the accumulator with 07EH, which is the opening flag. When a match occurs, the CPU identifies this as byte boundary, and thereafter processes the information byte-by-byte.

The CPU calls the RCV8 subroutine to get each byte into the accumulator. The CPU performs compare operations on (successively) the station address, the control field (which contains the RR response), and the closing flag. If any of these do not compare, the program jumps to the ERROR loop. If no error is found, the program jumps to the DONE loop.

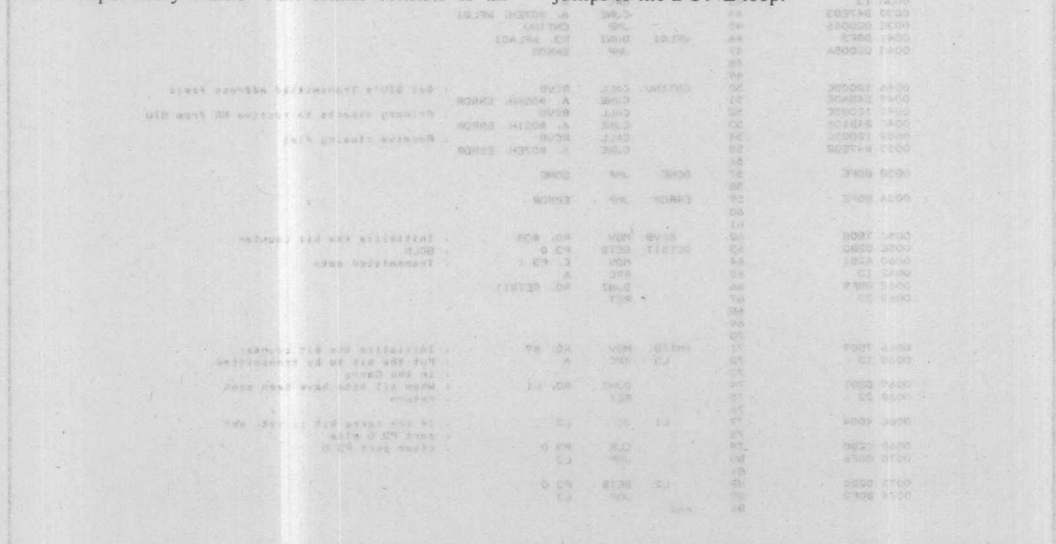


Figure 18-17. Loop-Back Mode Software



MCS-51 MACRO ASSEMBLER DATA			
ISIS-11 MCS-51 MACRO ASSEMBLER V2.0 OBJECT MODULE PLACED IN F1:DATA.OBJ ASSEMBLER INVOKED BY asm51 f1: data.man device(44)			
LOC	OBJ	LINE	SOURCE
		1	
		2	
0000	75C800	3	INIT: MOV STS, #00H ; Enable self test mode
0003	C2B1	4	CLR P3.1 ; Enable CTS
0005	C297	5	CLR P1.7 ; Initialize address
0007	75CEBA	6	MOV STAD, #BAH ; Initialize address
		7	
		8	CONFIGURE RECEIVE OPERATION
		9	
000A	75D86A	10	MOV NSNR, #6AH ; NS(S)=3, SES=0, NR(S)=5, SER=0
000D	75C901	11	MOV SPD, #01H ; NFCS=1
0010	75C8C2	12	MOV STS, #0C2H ; TBF=1, RBE=1, AM=1
		13	
		14	TRANSMIT A SUPERVISORY FRAME FROM THE PRIMARY STATION WITH THE POLL BIT SET AND A RNR COMMAND
		15	
		16	
0013	747E	17	SEND: MOV A, #7EH ; The SIU receives a flag first
0015	120066	18	CALL XMITB ; The address is next
0018	74BA	19	MOV A, #BAH ; RNR SUP FRAME with P/F=1, NR(P)=4
001A	120066	20	CALL XMITB ; Receive closing flag
001D	7495	21	MOV A, #095H ; Generate extra SCLK's to
001F	120066	22	CALL XMITB ; Initiate receive action
0022	747E	23	MOV A, #7EH ; Check for appropriate status
0024	120066	24	CALL XMITB ; Check for appropriate status
0027	D2B0	25	SETB P3.0 ; Check for appropriate status
0029	D2B0	26	SETB P3.0 ; Check for appropriate status
		27	
002B	E5CB	28	MOV A, STS ; Check for appropriate status
002D	B4722A	29	CJNE A, #72H, ERROR ; Check for appropriate status
		30	
		31	PREPARE TO RECEIVE RUP1'S RESPONSE TO PRIMARY'S RNR
		32	
		33	
		34	
0030	C2CC	35	RCV: CLR SI ; Clear SI
0032	7400	36	MOV A, #00H ; Clear ACC
0034	7B0C	37	MOV R3, #12 ; Try 12 times
		38	
		39	LOOK FOR THE OPENING FLAG
		40	
0036	D2B0	41	WFLAG1: SETB P3.0 ; SCLK
0038	A2B1	42	RRC A ; Transmitted data
003A	13	43	RRC A ; Transmitted data
003B	B47E03	44	CJNE A, #07EH, WFLG1
003E	020046	45	JMP CONTINU
0041	D8F3	46	WFLG1: DJNZ R3, WFLAG1
0043	02005A	47	JMP ERROR
		48	
		49	
0046	12005C	50	CONTINU: CALL RCVB ; Get SIU's Transmitted address field
0049	B4BA0E	51	CJNE A, #0BAH, ERROR ; Primary expects to receive RR from SIU
004C	12005C	52	CALL RCVB ; Receive closing flag
004F	B4B10B	53	CJNE A, #0B1H, ERROR ; Receive closing flag
0052	12005C	54	CALL RCVB ; Receive closing flag
0055	B47E02	55	CJNE A, #07EH, ERROR ; Receive closing flag
		56	
0058	80FE	57	DONE: JMP DONE
005A	80FE	58	ERROR: JMP ERROR
		59	
		60	
		61	
005C	7B0B	62	RCVB: MOV RO, #0B ; Initialize the bit counter
005E	D2B0	63	GETBIT: SETB P3.0 ; SCLK
0060	A2B1	64	MOV C, P3.1 ; Transmitted data
0062	13	65	RRC A ; Transmitted data
0063	D8F9	66	RRC A ; Transmitted data
0065	22	67	DJNZ RO, GETBIT
		68	RET
		69	
		70	
0066	7B09	71	XMITB: MOV RO, #9 ; Initialize the bit counter
0068	13	72	L3: RRC A ; Put the bit to be transmitted
0069	DB01	73	in the Carry
006B	22	74	DJNZ RET ; When all bits have been sent
		75	return
		76	
006C	4004	77	L1: JC L2 ; If the carry bit is set, set
006E	C2B0	78	CLR P3.0 ; port P3.0 else
0070	80F6	79	JMP L3 ; clear port P3.0
		80	
0072	D2B0	81	L2: SETB P3.0
0074	80F2	82	JMP L3
		83	
		84	end

Figure 18-17. Loop-Back Mode Software





# CHAPTER 19

## 8044 APPLICATION EXAMPLES

### 19.0 8044 APPLICATIONS EXAMPLES

#### 19.1 INTERFACING THE 8044 TO A MICROPROCESSOR

The 8044 is designed to serve as an intelligent controller for remote peripherals. However, it can also be used as an intelligent HDLC/SDLC front end for a microprocessor, capable of extensively off-loading link control functions for the CPU. In some applications, the 8044 can even be used for communications preprocessing, in addition to data link control.

This section describes a sample hardware interface for attaching the 8044 to an 8088. It is general enough to be extended to other microprocessors such as the 8086 or the 80186.

#### OVERVIEW

A sample interface is shown in Figure 19-1. Transmission occurs when the 8088 loads a 64 byte block of memory with some known data. The 8088 then enables the 8237A to DMA this data to the 8044. When the 8044 has received all of the data from the 8237A, it sends the data in a SDLC frame. The frame is captured by the Spectron Datascope®\* which displays it on a CRT in hex format.

In reception, the Datascope sends an SDLC information frame to the 8044. The 8044 receives the SDLC frame, buffers it, and sends it to the 8088's memory. In this example the 8044 is being operated in the NON-AUTO mode; therefore, it does not need to be polled by a primary station in order to transmit.

#### THE INTERFACE

The 8044 does not have a parallel slave port. The 8044's 32 I/O lines can be configured as a local microprocessor bus master. In this configuration, the 8044 can expand the ROM and RAM memory, control peripherals, and communicate with a microprocessor.

The 8044, like the 8051, does not have a Ready line, so there is no way to put the 8044 in wait state. The clock on the 8044 cannot be stopped. Dual port RAM could still be used, however, software arbitration would be the only way to prevent collisions. Another way to interface the 8044 with another CPU is to put a FIFO or queue between the two processors, and this was the method chosen for this design.

Figure 19-2 shows the schematic of the 8044/8088 interface. It involves two 8 bit tri-state latches, two SR flip-flops, and some logic gates (6 TTL packs). The circuitry implements a one byte FIFO. RS422 transceivers are used, which can be connected to a multidrop link. Figure 19-3 shows the 8088 and support circuitry; the memory and decoders are not shown. It is a basic 8088 Min Mode

system with an 8237A DMA controller and an 8259A interrupt controller.

DMA Channel One transfers a block of memory to the tri-state latch, while Channel Zero transfers a block of data from the latch to 8088's memory. The 8044's Interrupt 0 signal vectors the CPU into a routine which reads from the internal RAM and writes to the latch. The 8044's Interrupt 1 signal causes the chip to read from the latch and write to its on-chip data RAM. Both DMA requests and acknowledges are active low.

Initially, when the power is applied, a reset pulse coming from the 8284A initializes the SR flip-flops. In this initialization state, the 8044's transmit interrupt and the 8088's transmit DMA request are active; however, the software keeps these signals disabled until either of the two processors are ready to transmit. The software leaves the receive signals enabled, unless the receive buffers are full. In this way either the 8088 or the 8044 are always ready to receive, but they must enable the transmit signal when they have prepared a block to transmit. After a block has been transmitted or received, the DMA and interrupt signals return to the initial state.

The receive and transmit buffer sizes for the blocks of data sent between the 8044 and the 8088 have a maximum fixed length. In this case the buffer size was 64 bytes. The buffer size must be less than 192 bytes to enable 8044 to buffer the data in its on-chip RAM. This design allows blocks of data that are less than 64 bytes, and accommodates networks that allow frames of varying size. The first byte transferred between the 8088 and the 8044 is the byte count to follow; thus the 8044 knows how many bytes to receive before it transmits the SDLC frame. However, when the 8044 sends data to the 8088's memory, the 8237A will not know if the 8044 will send less than the count the 8237A was programmed for. To solve this problem, the 8237A is operated in the single mode. The 8044 uses an I/O bit to generate an interrupt request to the 8259A. In the 8088's interrupt routine, the 8237A's receive DMA channel is disabled, thus allowing blocks of data less than 64 bytes to be received.

#### THE SOFTWARE

The software for the 8044 and the 8088 is shown in Table 19-1. The 8088 software was written in PL/M86, and the 8044 software was written in assembly language.

The 8044 software begins by initializing the stack, interrupt priorities, and triggering types for the interrupts. At this point, the SIU parameter registers are initialized. The receive and transmit buffer starting addresses and lengths are loaded for the on-chip DMA. This DMA is for the serial port. The serial station address and the transmit control bytes are loaded too.

\*Datascope is a trademark of Spectron Inc.



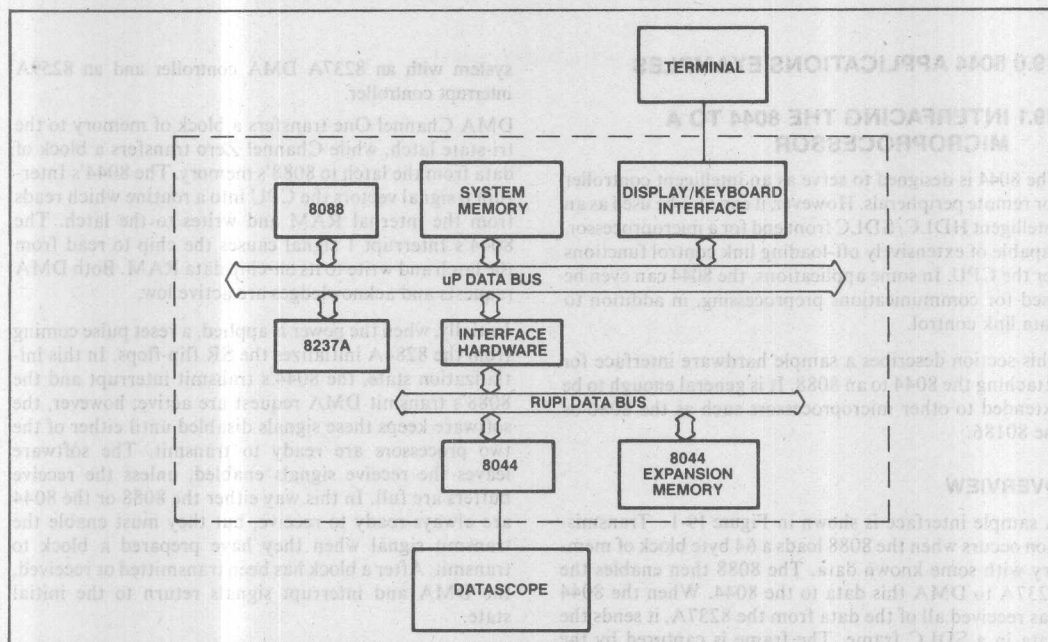


Figure 19-1. Block Diagram of 8088/8044 Interface Test

Once the initialization has taken place, the SIU interrupt is enabled, and the external interrupt which receives bytes from the 8088 is enabled. Setting the 8044's Receive Buffer Empty (RBE) bit enables the receiver. If this bit is reset, no serial data can be received. The 8044 then waits in a loop for either RECEIVE DMA interrupt or the SERIAL INT interrupt.

The RECEIVE DMA interrupt occurs when the 8237A is transferring a block of data to the 8044. The first time this interrupt occurs, the 8044 reads the latch and loads the count value into the R2 register. On subsequent interrupts, the 8044 reads the latch, loads the data into the transmit buffer, and decrements R2. When R2 reaches zero, the interrupt routine sends the data in an SDLC frame, and disables the RECEIVE DMA interrupt. After the frame has been transmitted, a serial interrupt is generated. The SERIAL INT routine detects that a frame has been transmitted and re-enables the RECEIVE DMA interrupt. Thus, while the frame is being transmitted through the SIU, the 8237A is inhibited from sending data to the 8044's transmit buffer.

The TRANSMIT DMA routine sends a block of data from the 8044's receive buffer to the 8088's memory. Normally this interrupt remains disabled. However, if a serial interrupt occurs, and the SERIAL INT routine detects that a frame has been received, it calls the SEND subroutine. The SEND subroutine loads the number of bytes which were received in the frame into the receive buffer. Register R1 points to the receive buff-

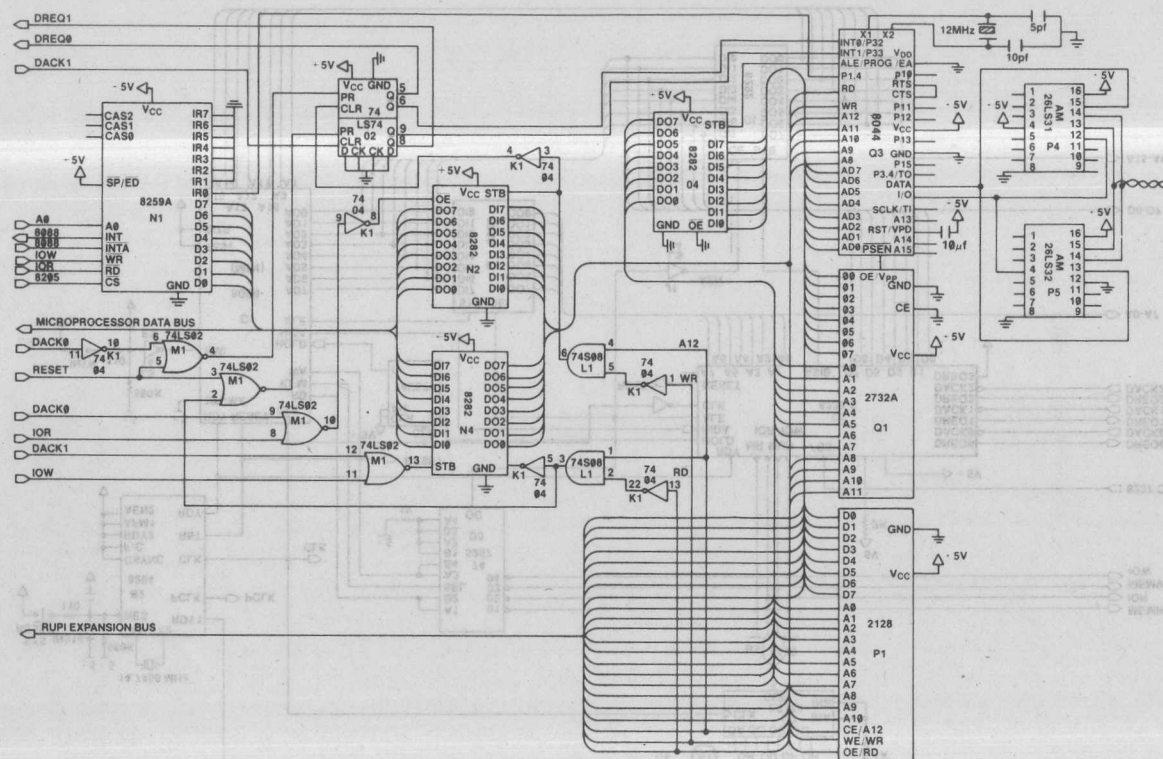
er and R2 is loaded with the count. The TRANSMIT DMA interrupt is enabled, and immediately upon returning from the SERIAL INT routine, the interrupt is acknowledged. Each time the TRANSMIT DMA interrupt occurs, a byte is read from the receive buffer, written to the latch, and R2 is decremented. When R2 reaches 0, the TRANSMIT DMA interrupt is disabled, the SIU receiver is re-enabled, and the 8044 interrupts the 8088.

The 8088 software simply transmits a block of data and receives a block of data, then stops. The software begins by initializing the 8237A, and the 8259A. It then loads a block of memory with some data and enables the 8237A to transmit the data. In the meantime the 8088 waits in a loop. After a block of data is received from the 8044, the 8088 is interrupted, and it shuts off the 8237A receive DMA.

## CONCLUSION

For the software shown in Table 19-1, the transfer rate from the 8088's memory to the 8044 was measured at 75K bytes/sec. This transfer rate largely depends upon the number of instructions in the 8044's interrupt service routine. Fewer instructions result in a higher transfer rate.

There are many ways of interfacing the 8044 locally to another microprocessor: FIFO's, dual port RAM with software arbitration, and 8255's are just a few. Alternative approaches, which may be more optimal for certain applications, are certainly possible.



**Figure 19-2. 8044 Interface to the 8088**

### Figure 19-3. 8088 Min Mode System

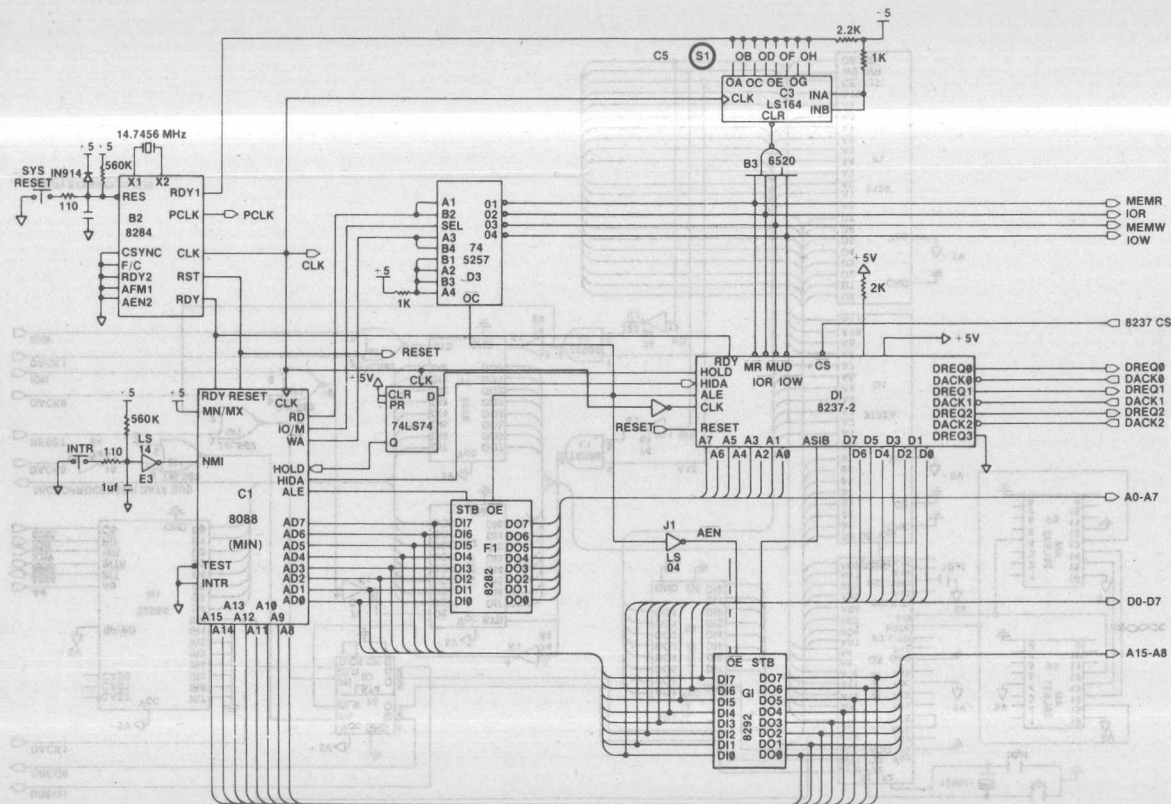


Table 19-1. Transmit and Receive Software for an 8044/8088 System

LOC	OBJ	LINE	SOURCE
		1	Sdebug title (8044/8088 INTERFACE)
		2	
0000		3	
		4	FIRST_BYTE BIT 0 ; FLAG
		5	
0000		6	ORG 0
0000 8024		7	SJMP INIT
		8	
0026		9	ORG 26H
		10	
0026 7581AA		11	INIT: MOV SP, #170 ; INITIALIZE STACK
0029 75B800		12	MOV IP, #00 ; ALL INTERRUPTS ARE EQUAL PRIORITY
002C 75C954		13	MOV SMD, #54H ; TIMER 1 OVERFLOW, NRZI, PRE-FRAME SYNC
002F 758844		14	MOV TCON, #44H ; EDGE TRIGGERED EXTERNAL INTERRUPT 1
		15	; LEVEL TRIGGERED EXTERNAL INTERRUPT 0
		16	; TIMER 1 ON
0032 758DEC		17	MOV TH1, #0ECH ; INITIALIZE TIMER, 3125 BPS
0035 758920		18	MOV TMOD, #20H ; TIMER 1 AUTO RELOAD
		19	
0038 75DC6A		20	MOV TBS, #106 ; SET UP SIU PARAMETER REGISTERS
003B 75DB40		21	MOV TBL, #64
003E 75CC2A		22	MOV RBS, #42
0041 75CB40		23	MOV RBL, #64
0044 75CE55		24	MOV STAD, #55H
0047 75DA11		25	MOV TCB, #00010001B ; RR, P/F=1
		26	
004A 901000		27	MOV DPTR, #1000H ; DPTR POINTS TO TRI-STATE LATCH
004D D200		28	SETB FIRST_BYTE ; FLAG TO INDICATE FIRST BYTE
		29	; FOR RECEIVE INTERRUPT ROUTINE
004F D2CE		30	SETB RBE ; READY TO RECEIVE
0051 75A894		31	MOV IE, #10010100B ; ENABLE RECEIVE DMA AND SIU INTERRUPT
		32	
0054 80FE		33	SJMP \$ ; WAIT HERE FOR INTERRUPTS
		34	
0056 80FE		35	ERROR: SJMP ERROR
		36	+1 \$EJ
		37	***** SUBROUTINES *****
		38	
0058 85CD29		39	SEND: MOV 41, RFL ; FIRST BYTE IN BLOCK IS COUNT
005B 7929		40	MOV R1, #41 ; POINT TO BLOCK OF DATA
005D AACD		41	MOV R2, RFL ; LOAD COUNT
005F 0A		42	INC R2
0060 D2A8		43	SETB EX0 ; ENABLE DMA TRANSMIT INTERRUPT
0062 22		44	RET
		45	
		46	
		47	
		48	***** INTERRUPT SERVICE ROUTINES *****
		49	
0063		50	LOC_TMPSET \$ ; SET UP INTERRUPT TABLE JUMP
0013		51	ORG 0013H
0013 020063		52	LJMP RECEIVE_DMA
0063		53	ORG LOC_TMP
		54	
		55	RECEIVE_DMA:



0063	10000E	56	JBC	FIRST_BYTE, L1	; THE FIRST BYTE TRANSFERRED IS THE COUNT
		57			
		58			
0066	E0	59	MOVX	A, @DPTR	; READ THE LATCH
0067	F6	60	MOV	@R0, A	; PUT IT IN TRANSMIT BUFFER
0068	08	61	INC	R0	
0069	DA08	62	DJNZ	R2, L2	; AFTER READING BYTES,
		63			
006B	D2CF	64	SETB	TBF	; SEND DATA
006D	D2CD	65	SETB	RTS	
006F	D200	66	SETB	FIRST_BYTE	
0071	C2AA	67	CLR	EX1	
		68			
0073	32	69	L2: RETI		
		70			
0074	786A	71	L1: MOV	R0, #106	; R0 IS A POINTER TO THE TRANSMIT
		72			; BUFFER STARTING ADDRESS
0076	E0	73	MOVX	A, @DPTR	; PUT THE FIRST BYTE INTO
0077	FA	74	MOV	R2, A	; R2 FOR THE COUNT
0078	32	75	RETI		
		76			
0079		77	LOC_TMPSET	\$	
0003		78	ORG	0003H	
0003	020079	79	LJMP	TRANSMIT_DMA	
0079		80	ORG	LOC_TMP	
		81			
		82	TRANSMIT_DMA		
		83			
0079	E7	84	MOV	A, @R1	; READ BYTE OUT OF THE RECEIVE BUFFER
007A	F0	85	MOVX	@DPTR, A	; WRITE IT TO THE LATCH
007B	09	86	INC	R1	
007C	DA08	87	DJNZ	R2, L3	; WHEN ALL BYTES HAVE BEEN SENT
		88			
007E	C2A8	89	CLR	IE. 0	; DISABLE INTERRUPT
0080	C294	90	CLR	PI. 4	; CAUSE 8088 INTERRUPT TO TERMINATE DMA
0082	D294	91	SETB	PI. 4	
0084	D2CE	92	SETB	RBE	; ENABLE RECEIVER AGAIN
		93			
0086	32	94	L3: RETI		
		95			
		96			
		97			
0087		98	LOC_TMPSET	\$	
0023		99	ORG	0023H	
0023	020087	100	LJMP	SERIAL_INT	
0087		101	ORG	LOC_TMP	
		102			
		103	SERIAL_INT:		
		104			
0087	30CE06	105	JNB	RBE, RCV	; WAS A FRAME RECEIVED
008A	30CF0B	106	JNB	TBF, XMIT	; WAS A FRAME TRANSMITTED
008D	020056	107	LJMP	ERROR	; IF NEITHER ERROR
		108			
0090	20CBC3	109	RCV: JB	BOV, ERROR	; IF BUFFER OVERRUN THEN ERROR
0093	1158	110	CALL	SEND	; SEND THE FRAME TO THE 8088
0095	C2CC	111	CLR	SI	
0097	32	112	RETI		
		113			
0098	C2CC	114	XMIT: CLR	SI	

```
009A D2AA 115 SETB EX1
009C 32 116 RETI
117
118 END
```

# SYMBOL TABLE LISTING

NAME	TYPE	VALUE	ATTRIBUTES
BOV	B ADDR	00C8H.3 A	
ERROR	C ADDR	0056H A	
EX0	B ADDR	00A8H.0 A	
EX1	B ADDR	00A8H.2 A	
FIRST_BYTE	B ADDR	0020H.0 A	
IE	D ADDR	00A8H A	
INIT	C ADDR	0026H A	
IP	D ADDR	00B8H A	
L1	C ADDR	0074H A	
L2	C ADDR	0073H A	
L3	C ADDR	0086H A	
LOC_TMP	C ADDR	0087H A	
P1	D ADDR	0090H A	
RBE	B ADDR	00C8H.6 A	
RBL	D ADDR	00CBH A	
RBS	D ADDR	00CCH A	
RCV	C ADDR	0090H A	
RECEIVE_DMA	C ADDR	0063H A	
RFL	D ADDR	00CDH A	
RTS	B ADDR	00C8H.5 A	
SEND	C ADDR	0058H A	
SERIAL_INT	C ADDR	0087H A	
SI	B ADDR	00C8H.4 A	
SMD	D ADDR	00C9H A	
SP	D ADDR	0081H A	
STAD	D ADDR	00CEH A	
TBF	B ADDR	00C8H.7 A	
TBL	D ADDR	00DBH A	
TBS	D ADDR	00DCH A	
TCB	D ADDR	00DAH A	
TCON	D ADDR	0088H A	
TH1	D ADDR	008DH A	
TMOD	D ADDR	0089H A	
TRANSMIT_DMA	C ADDR	0079H A	
XMIT	C ADDR	0098H A	

REGISTER BANK(S) USED: 0, TARGET MACHINE(S): 8044

ASSEMBLY COMPLETE, NO ERRORS FOUND.

Table 19-2. PL/M-86 Compiler Rupi/8088 Interface Example

SERIES-III PL/M-86 V1.0 COMPILATION OF MODULE RUPI\_88  
 OBJECT MODULE PLACED IN : F1:R88.OBJ  
 COMPILER INVOKED BY: PLM86.86 : F1:R88.SRC

```

$DEBUG
$TITLE ('RUPI/8088 INTERFACE EXAMPLE')

1      RUPI_88: DO;
2      1      DECLARE
          LIT      LITERALLY 'LITERALLY',
          TRUE     LIT      '01H',
          FALSE    LIT      '00H',
          RECV_BUFFER(64) BYTE,
          XMIT_BUFFER(64) BYTE,
          I         BYTE,
          WAIT      BYTE,
          /* 8237 PORTS */
          MASTER_CLEAR_37 LIT      'OFFDDH',
          COMMAND_37      LIT      'OFFDBH',
          ALL_MASK_37     LIT      'OFFDFH',
          SINGLE_MASK_37  LIT      'OFFDAH',
          STATUS_37       LIT      'OFFDBH',
          REQUEST_REG_37   LIT      'OFFD9H',
          MODE_REG_37      LIT      'OFFDBH',
          CLEAR_BYTE_PTR_37 LIT      'OFFDCH',
          CH0_ADDR         LIT      'OFFDOH',
          CH0_COUNT        LIT      'OFFD1H',
          CH1_ADDR         LIT      'OFFD2H',
          CH1_COUNT        LIT      'OFFD3H',
          CH2_ADDR         LIT      'OFFD4H',
          CH2_COUNT        LIT      'OFFD5H',
          CH3_ADDR         LIT      'OFFD6H',
          CH3_COUNT        LIT      'OFFD7H',
          /* 8237 BIT ASSIGNMENTS */
          CH0_SEL          LIT      '00H',
          CH1_SEL          LIT      '01H',
          CH2_SEL          LIT      '02H',
          CH3_SEL          LIT      '03H',
          WRITE_XFER       LIT      '04H',
          READ_XFER        LIT      '08H',
          DEMAND_MODE       LIT      '00H',
          SINGLE_MODE       LIT      '40H',
          BLOCK_MODE        LIT      '80H',
          SET_MASK          LIT      '04H',
          $EJECT
          /* 8259 PORTS */
          STATUS_POLL_59   LIT      'OFFEOH',
          ICW1_59          LIT      'OFFEOH',
          OCW1_59          LIT      'OFFE1H',
          OCW2_59          LIT      'OFFEOH',
          OCW3_59          LIT      'OFFEOH',
          ICW2_59          LIT      'OFFE1H',
          ICW3_59          LIT      'OFFE1H',
          ICW4_59          LIT      'OFFE1H',
          /* INTERRUPT SERVICE ROUTINE */
3      1      OFF_RECV_DMA: PROCEDURE INTERRUPT 32;
4      2      OUTPUT(SINGLE_MASK_37)=40H;
5      2      WAIT=FALSE;
6      2      END;
  
```

# THE RUP1™-44

```

7 1      DISABLE;

/* INITIALIZE 8237 */

8 1      OUTPUT(MASTER_CLEAR_37)    =0;
9 1      OUTPUT(COMMAND_37)         =040H;
10 1     OUTPUT(ALL_MASK_37)        =0FH;
11 1     OUTPUT(MODE_REG_37)        =(SINGLE_MODE OR WRITE_XFER OR CHO_SEL);
12 1     OUTPUT(MODE_REG_37)        =(SINGLE_MODE OR READ_XFER OR CH1_SEL);
13 1     OUTPUT(CLEAR_BYTE_PTR_37)  =0;
14 1     OUTPUT(CHO_ADDR)           =00H;
15 1     OUTPUT(CHO_COUNT)          =44H;
16 1     OUTPUT(CHO_COUNT)          =64;
17 1     OUTPUT(CHO_COUNT)          =00;
18 1     OUTPUT(CH1_ADDR)           =40H;
19 1     OUTPUT(CH1_ADDR)           =40H;
20 1     OUTPUT(CH1_COUNT)          =64;
21 1     OUTPUT(CH1_COUNT)          =00;

/* INITIALIZE 8259 */

22 1     OUTPUT(ICW1_59)            =13H; /*SINGLE MODE, EDGE TRIGGERED
/* INPUT, 8086 INTERRUPT TYPE*/
23 1     OUTPUT(ICW2_59)            =20H; /*INTERRUPT TYPE 32*/
24 1     OUTPUT(ICW4_59)            =03H; /*AUTO-EOI*/
25 1     OUTPUT(OCW1_59)            =0FEH; /*ENABLE INTERRUPT LEVEL 0*/

26 1     *EJECT
27 1     CALL SET*INTERRUPT (32,OFF_RECV_DMA); /*LOAD INTERRUPT VECTOR LOCATION*/
28 1     XMIT_BUFFER(0)=64; /*THE FIRST BYTE IN THE BLOCK OF DATA IS THE NUMBER
/* OF BYTES TO BE TRANSFERRED; NOT INCLUDING THE FIRST BYTE*/
29 1     DO I= 1 TO 64; /* FILL UP THE XMIT_BUFFER WITH DATA */
30 2     XMIT_BUFFER(I)=I;
31 2     END;
32 1     OUTPUT(ALL_MASK_37)=0FCH; /*ENABLE CHANNEL 1 AND 2 */
33 1     ENABLE;
34 1     WAIT=TRUE;
35 1     DO WHILE WAIT;
36 2     END; /* A BLOCK OF DATA WILL BE TRANSFERRED TO THE RUP1.
/* WHEN THE RUP1 RECEIVES A BLOCK OF DATA IT WILL
/* SEND IT TO THE 8086 MEMORY AND INTERRUPT THE 8086.
/* THE INTERRUPT SERVICE ROUTINE WILL SHUT OFF THE DMA
/* CONTROLLER AND SET 'WAIT' FALSE */
37 1     DO WHILE 1;
38 2     END;

MODULE INFORMATION:
CODE AREA SIZE      = 00D7H 215D
CONSTANT AREA SIZE  = 0000H 0D
VARIABLE AREA SIZE  = 00B2H 130D
MAXIMUM STACK SIZE  = 001EH 30D
124 LINES READ
0 PROGRAM WARNINGS
0 PROGRAM ERRORS
END OF PL/M-86 COMPILATION

```



# A HIGH PERFORMANCE NETWORK USING THE 8044

## 19.2.1 Introduction

This section describes the design of an SDLC data link using the 8044 (RUP1) to implement a primary station and a secondary station. The design was implemented and tested. The following discussion assumes that the reader understands the 8044 and SDLC. This section is divided into two parts. First the data link design example is discussed. Second the software modules used to implement the data link are described. To help the reader understand the discussion of the software, flow charts and software listings are displayed in Appendix A and Appendix B, respectively.

### Application Description

This particular data link design example uses a two wire half-duplex multidrop topology as shown in figure 19-4. In an SDLC multidrop topology the primary station communicates with each secondary station. The secondary stations communicate only to the primary. Because of this hierarchical architecture, the logical topology for an SDLC multidrop is a star as shown in figure 19-5. Although the physical topology of this data link is multidrop, the easiest way to understand the information flow is to think of the logical (star) topology. The term data link in this case refers to the logical communication pathways between the primary station and the secondary stations. The data links are shown in figure 19-5 as two way arrows.

The application example uses dumb async terminals to interface to the SDLC network. Each secondary station has an async terminal connected to it. The secondary stations are in effect protocol converters which allows any async terminal to communicate with any other async terminal on the network. The secondary stations use an 8044 with a UART to convert SDLC to async. Figure 19-6 displays a block diagram of the data link. The primary station, controls the data link. In addition to data link control the primary provides a higher level layer which is a path control function or networking layer. The primary serves as a message exchange or switch. It receives information from one secondary station and retransmits it to another secondary station. Thus a virtual end to end connection is made between any two secondary stations on the network.

Three separate software modules were written for this network. The first module is a Secondary Station Driver (SSD) which provides an SDLC data link interface and a user interface. This module is a general purpose driver which requires application software to run it. The user interface to the driver provides four functions: OPEN,

CLOSE, TRANSMIT, and SIU\_RECV. Using these four functions properly will allow any application software to communicate over this SDLC data link without knowing the details of SDLC. The secondary station driver uses the 8044's AUTO mode.

The second module is an example of application software which is linked to the secondary station driver. This module drives the 8251A, buffers data, and interfaces with the secondary station driver's user interface.

The third module is a primary station, which is a stand-alone program (i.e., it is not linked to any other module). The primary station uses the 8044's NON-AUTO or FLEXIBLE mode. In addition to controlling the data link it acts as a message switch. Each time a secondary station transmits a frame, it places the destination address of the frame in the first byte of the information or I field. When the primary station receives a frame, it removes the first byte in the I field and retransmits the frame to the secondary station whose address matches this byte.

This network provides two complete layers of the OSI (Open Systems Interconnection) reference model: the physical layer and the data link layer. The physical layer implementation uses the RS-422 electrical interface. The mechanical medium consists of ribbon cable and connectors. The data link layer is defined by SDLC. SDLC's use of acknowledgements and frame numbering guarantees that messages will be received in the same order in which they were sent. It also guarantees message integrity over the data link. However this network will not guarantee secondary to secondary message delivery, since there are acknowledgements between secondary stations.

## 19.2.2 Hardware

The schematic of the hardware is given in figure 19-7. The 8251A is used as an async communications controller, in support of the 8044. TxRDY and RxRDY on the 8251A are both tied to the two available external interrupts of the 8044 since the secondary station driver is totally interrupt driven. The 8044 buffers the data and some variables in a 2016 (2K x 8 static RAM). The 8254 programmable interval timer is employed as a programmable baud rate generator and system clock driver for the 8251A. The third output from the 8254 could be used as an external baud rate generator for the 8044. The 2732A shown in the diagram was not used since the software for both the primary and secondary stations used far less than the 4 Kbytes provided on the 8744. For the async interface, the standard RS-232

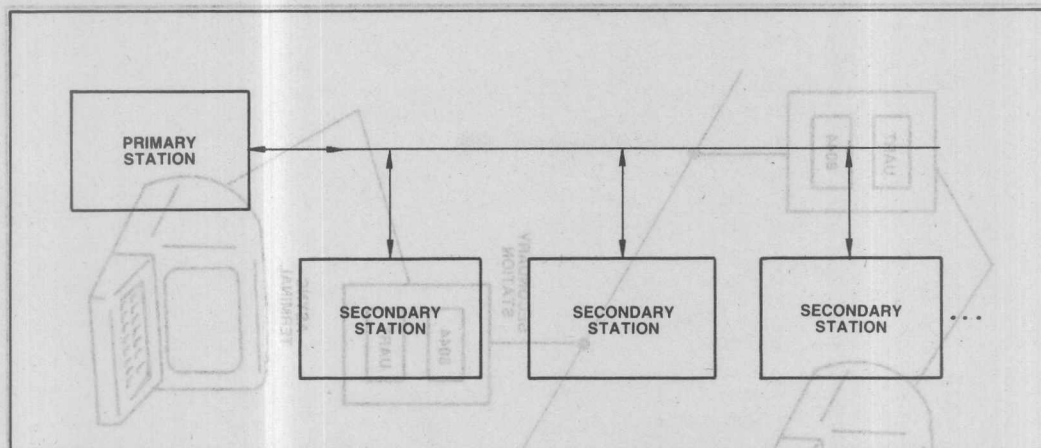


Figure 19-4. SDLC Multidrop Topology

mechanical and electrical interface was used. For the SDLC channel, a standard two wire three state RS-422 driver is used. A DIP switch connected to one of the available ports on the 8044 allows the baud rate, parity, and stop bits to be changed on the async interface. The primary station hardware does not use the USART, 8254, nor the RS-232 drivers.

### 19.2.3 SDLC Basic Repertoire

The SDLC commands and responses implemented in the data link include the SDLC Basic Repertoire as defined in the IBM SDLC General Information manual. Table 19-3 shows the commands and responses that the primary and the secondary station in this data link design recognize and send.

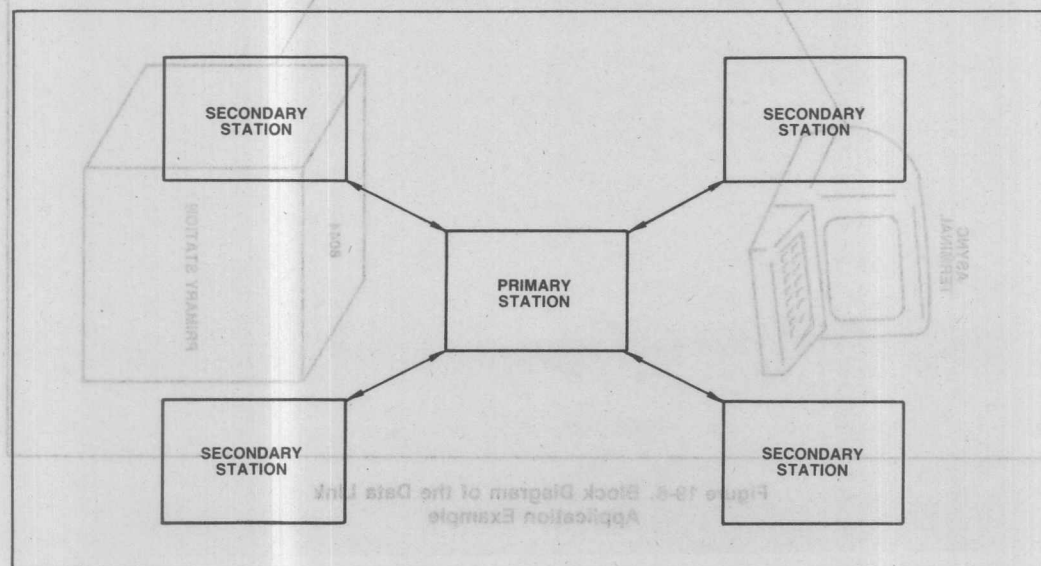


Figure 19-5. SDLC Logical Topology

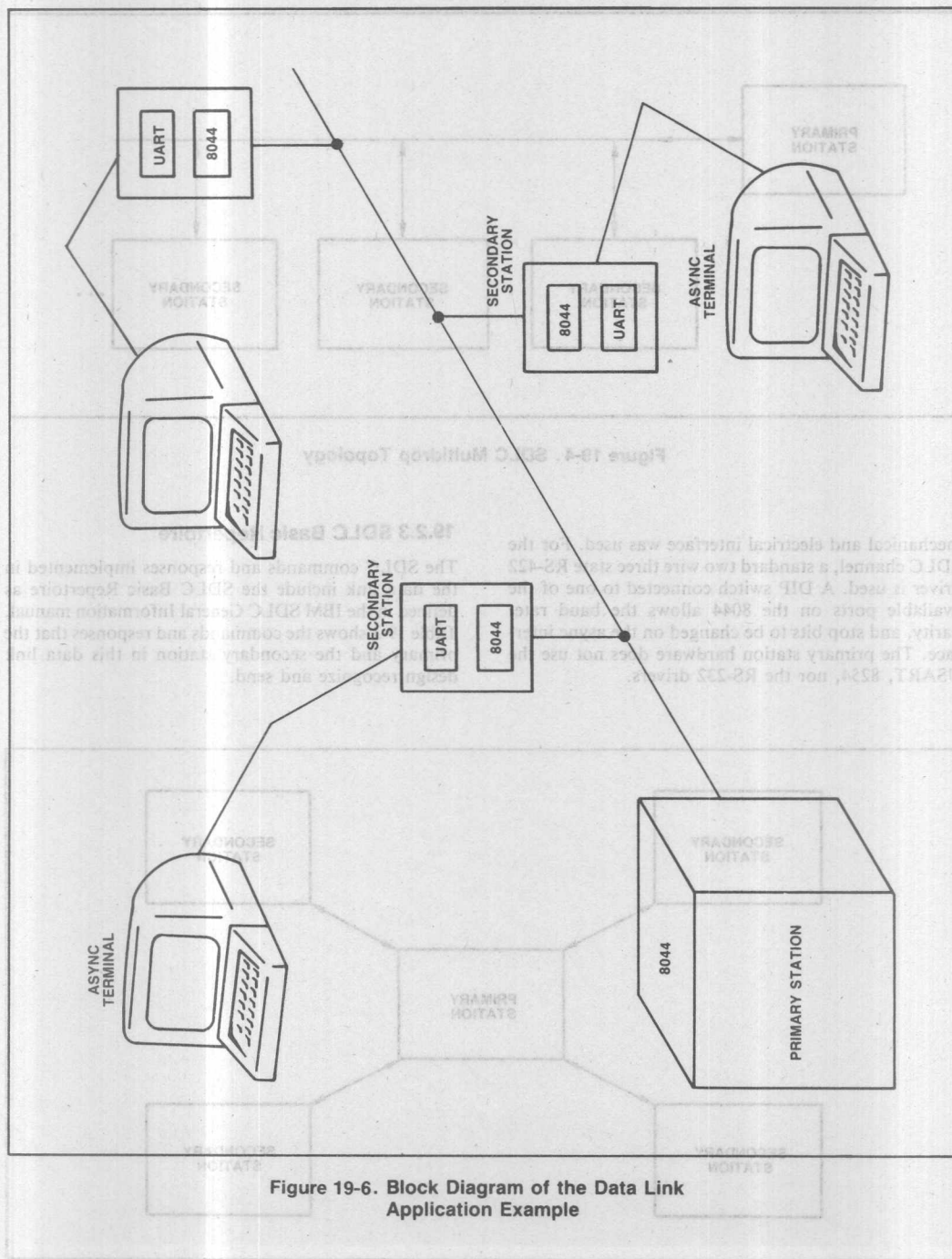


Figure 19-6. Block Diagram of the Data Link Application Example

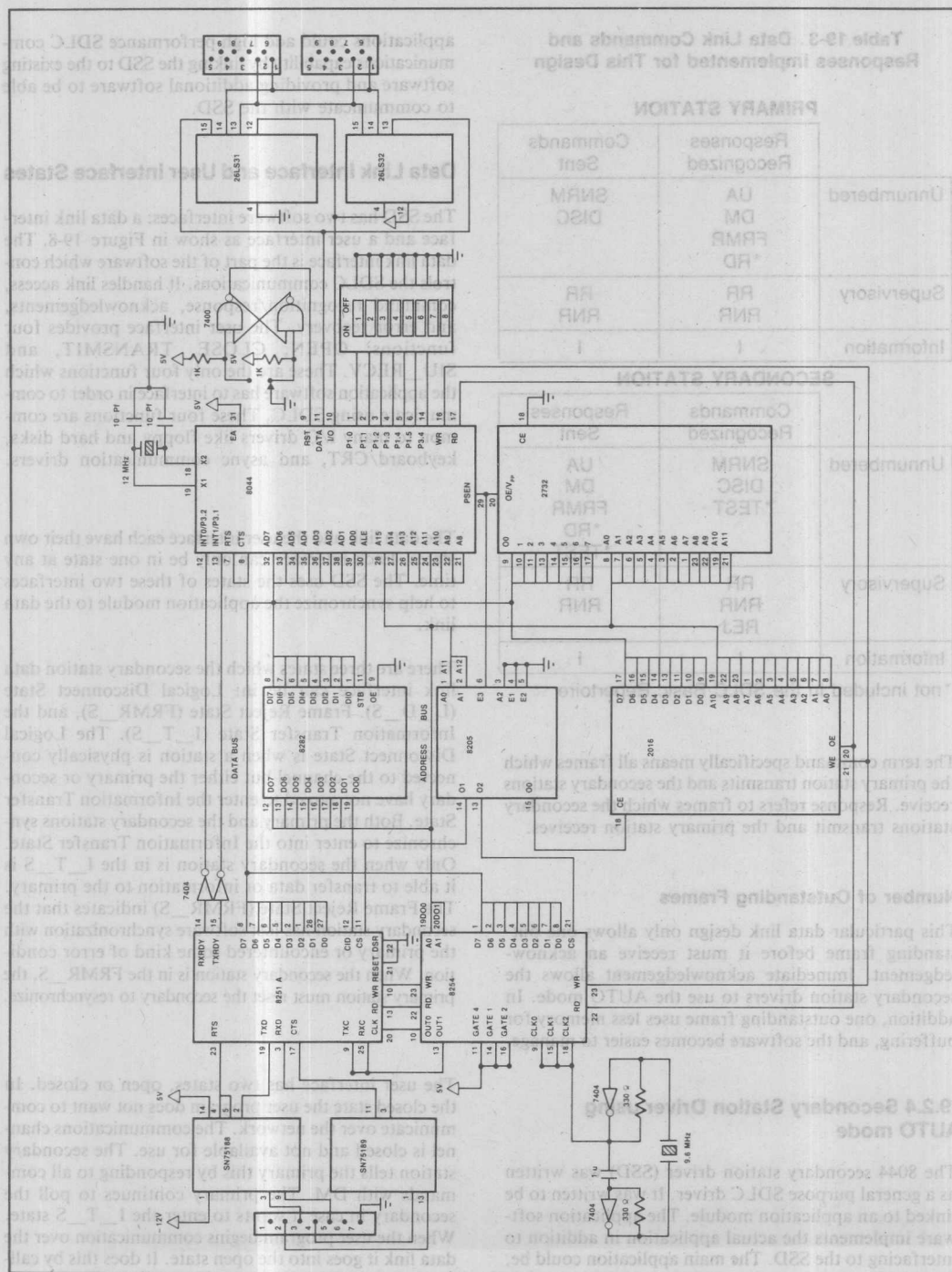


Figure 19-7. Schematic of Async/SDLC Secondary Station Protocol Converter



**Table 19-3. Data Link Commands and Responses Implemented for This Design**

PRIMARY STATION		
	Responses Recognized	Commands Sent
Unnumbered	UA DM FRMR *RD	SNRM DISC
Supervisory	RR RNR	RR RNR
Information	I	I

SECONDARY STATION		
	Commands Recognized	Responses Sent
Unnumbered	SNRM DISC *TEST	UA DM FRMR *RD *TEST
Supervisory	RR RNR REJ	RR RNR
Information	I	I

\*not included in the SDLC Basic Repertoire

The term command specifically means all frames which the primary station transmits and the secondary stations receive. Response refers to frames which the secondary stations transmit and the primary station receives.

#### Number of Outstanding Frames

This particular data link design only allows one outstanding frame before it must receive an acknowledgement. Immediate acknowledgement allows the secondary station drivers to use the AUTO mode. In addition, one outstanding frame uses less memory for buffering, and the software becomes easier to manage.

#### 19.2.4 Secondary Station Driver using AUTO mode

The 8044 secondary station driver (SSD) was written as a general purpose SDLC driver. It was written to be linked to an application module. The application software implements the actual application in addition to interfacing to the SSD. The main application could be, a printer or plotter, a medical instrument, or a terminal. The SSD is independent of the main application, it just provides the SDLC communications. Existing 8051

applications could add high performance SDLC communications capability by linking the SSD to the existing software and providing additional software to be able to communicate with the SSD.

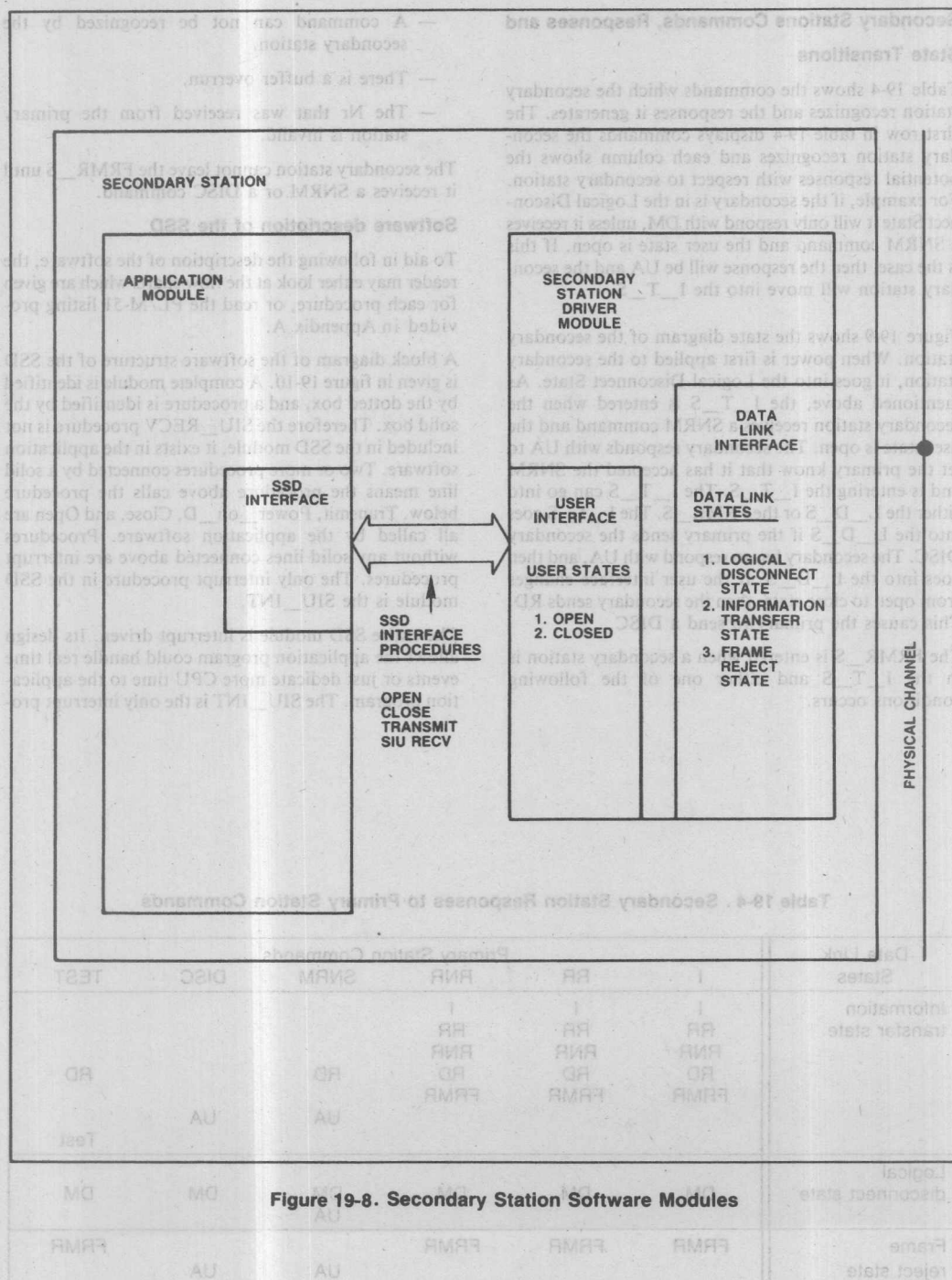
#### Data Link Interface and User Interface States

The SSD has two software interfaces: a data link interface and a user interface as show in Figure 19-8. The data link interface is the part of the software which controls the SDLC communications. It handles link access, command recognition/response, acknowledgements, and error recovery. The user interface provides four functions: OPEN, CLOSE, TRANSMIT, and SIU\_RECV. These are the only four functions which the application software has to interface in order to communicate using SDLC. These four functions are common to many I/O drivers like floppy and hard disks, keyboard/CRT, and async communication drivers.

The data link and the user interface each have their own states. Each interface can only be in one state at any time. The SSD uses the states of these two interfaces to help synchronize the application module to the data link.

There are three states which the secondary station data link interface can be in: Logical Disconnect State (L\_D\_S), Frame Reject State (FRMR\_S), and the Information Transfer State (I\_T\_S). The Logical Disconnect State is when a station is physically connected to the channel but either the primary or secondary have not agreed to enter the Information Transfer State. Both the primary and the secondary stations synchronize to enter into the Information Transfer State. Only when the secondary station is in the I\_T\_S is it able to transfer data or information to the primary. The Frame Reject State (FRMR\_S) indicates that the secondary station has lost software synchronization with the primary or encountered some kind of error condition. When the secondary station is in the FRMR\_S, the primary station must reset the secondary to resynchronize.

The user interface has two states, open or closed. In the closed state the user program does not want to communicate over the network. The communications channel is closed and not available for use. The secondary station tells the primary this by responding to all commands with DM. The primary continues to poll the secondary in case it wants to enter the I\_T\_S state. When the user program begins communication over the data link it goes into the open state. It does this by calling the OPEN procedure. When the user interface is in the open state it may transfer information to the primary.



## Secondary Stations Commands, Responses and State Transitions

Table 19-4 shows the commands which the secondary station recognizes and the responses it generates. The first row in table 19-4 displays commands the secondary station recognizes and each column shows the potential responses with respect to secondary station. For example, if the secondary is in the Logical Disconnect State it will only respond with DM, unless it receives a SNRM command and the user state is open. If this is the case, then the response will be UA, and the secondary station will move into the I\_T\_S.

Figure 19-9 shows the state diagram of the secondary station. When power is first applied to the secondary station, it goes into the Logical Disconnect State. As mentioned above, the I\_T\_S is entered when the secondary station receives a SNRM command and the user state is open. The secondary responds with UA to let the primary know that it has accepted the SNRM and is entering the I\_T\_S. The I\_T\_S can go into either the L\_D\_S or the FRMR\_S. The I\_T\_S goes into the L\_D\_S if the primary sends the secondary DISC. The secondary has to respond with UA, and then goes into the L\_D\_S. If the user interface changes from open to close state, then the secondary sends RD. This causes the primary to send a DISC.

The FRMR\_S is entered when a secondary station is in the I\_T\_S and either one of the following conditions occurs.

- A command can not be recognized by the secondary station.
- There is a buffer overrun.
- The Nr that was received from the primary station is invalid.

The secondary station cannot leave the FRMR\_S until it receives a SNRM or a DISC command.

## Software description of the SSD

To aid in following the description of the software, the reader may either look at the flow charts which are given for each procedure, or read the PL/M-51 listing provided in Appendix A.

A block diagram of the software structure of the SSD is given in figure 19-10. A complete module is identified by the dotted box, and a procedure is identified by the solid box. Therefore the SIU\_RECV procedure is not included in the SSD module, it exists in the application software. Two or more procedures connected by a solid line means the procedure above calls the procedure below. Transmit, Power\_on\_D, Close, and Open are all called by the application software. Procedures without any solid lines connected above are interrupt procedures. The only interrupt procedure in the SSD module is the SIU\_INT.

The entire SSD module is interrupt driven. Its design allows the application program could handle real time events or just dedicate more CPU time to the application program. The SIU\_INT is the only interrupt pro-

Table 19-4. Secondary Station Responses to Primary Station Commands

Data Link States	Primary Station Commands					
	I	RR	RNR	SNRM	DISC	TEST
Information transfer state	I	I	I			
	RR	RR	RR			
	RNR	RNR	RNR			
	RD	RD	RD	RD		RD
	FRMR	FRMR	FRMR			
				UA	UA	
						Test
Logical disconnect state	DM	DM	DM	DM	DM	DM
				UA		
Frame reject state	FRMR	FRMR	FRMR			FRMR
				UA	UA	

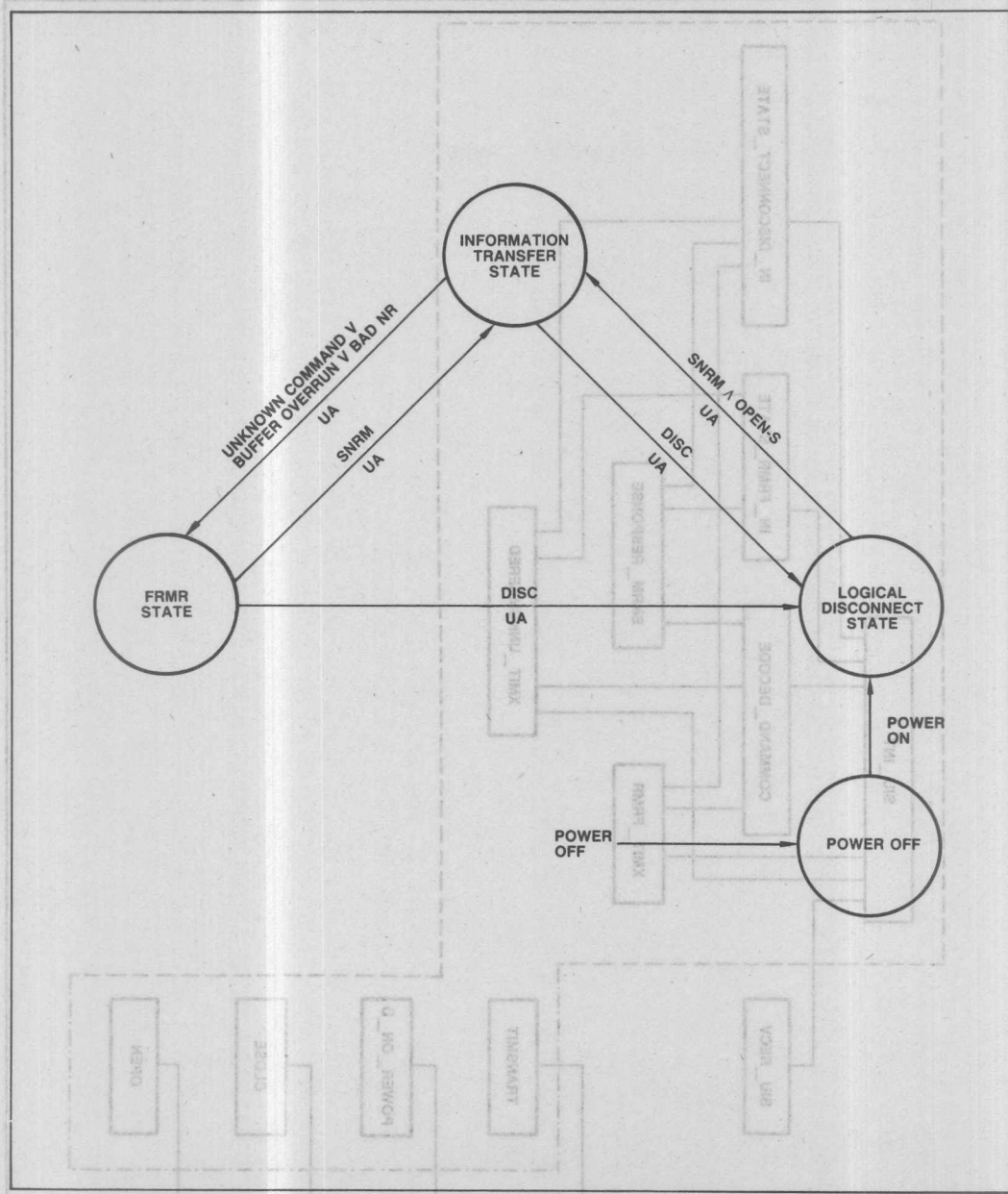


Figure 19-9. State Diagram of Secondary Station



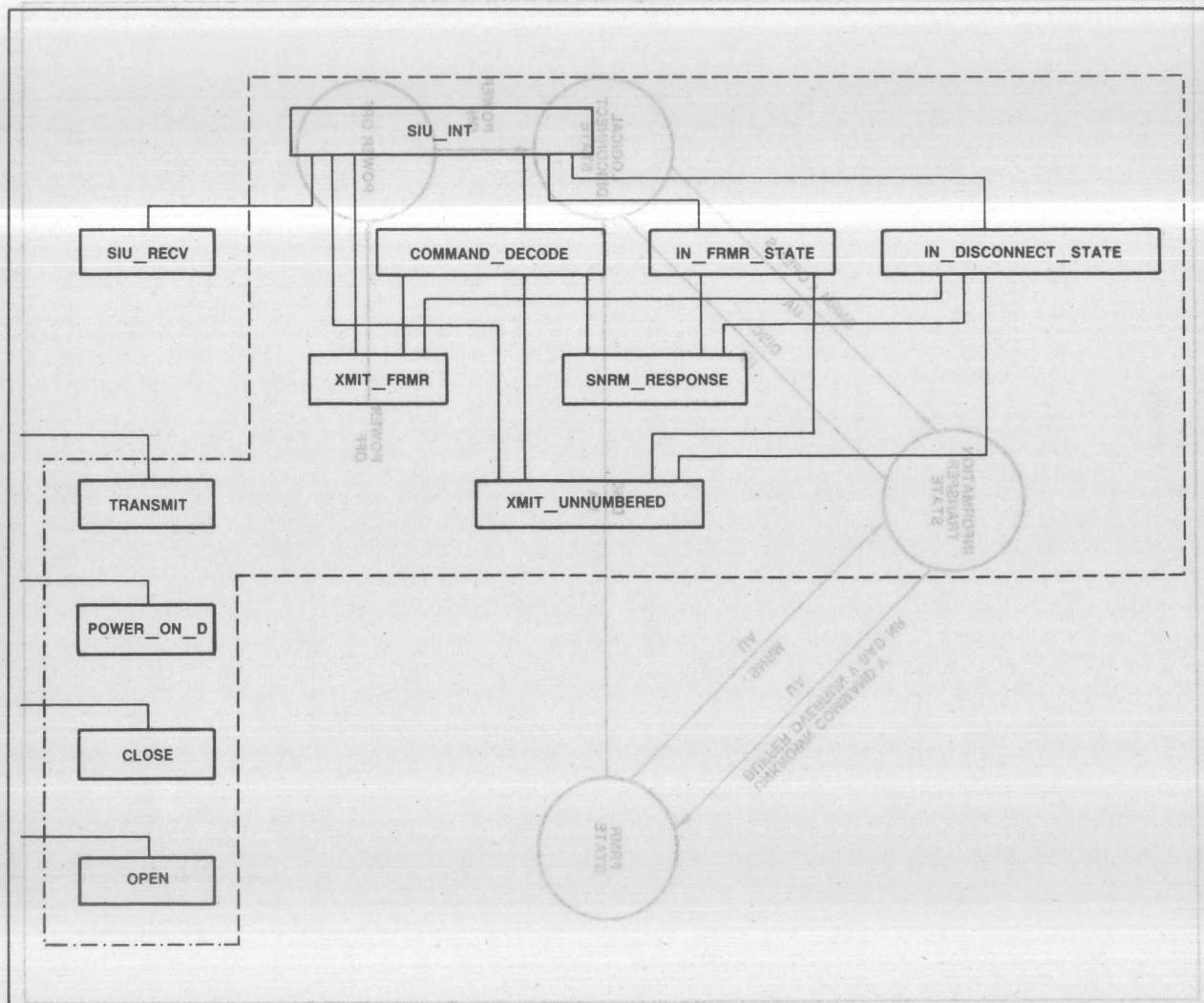


Figure 19-10. Secondary Station Driver

cedure in the SSD. It is automatically entered when an SIU interrupt occurs. This particular interrupt can be the lowest priority interrupt in the system.

### SSD Initialization

Upon reset the application software is entered first. The application software initializes its own variables then calls Power\_On\_D which is the SSD's initialization routine. The SSD's initialization sets up the transmit and receive data buffer pointers (TBS and RBS), the receive buffer length (RBL), and loads the State variables. The STATION\_STATE begins in the L\_D\_S state, and the USER\_STATE begins in the closed state. Finally Power\_On\_D initializes XMIT\_BUFFER\_EMPTY which is a bit flag. This flag serves as a semaphore between the SSD and the application software to indicate the status of the on chip transmit buffer. The SSD does not set the station address. It is the application software's responsibility to do this. After initialization, the SSD is ready to respond to all of the primary stations commands. Each time a frame is received with a matching station address and a good CRC, the SIU\_INT procedure is entered.

### SIU\_INT Procedure

The first thing the SIU\_INT procedure clears the serial interrupt bit (SI) in the STS register. If the SIU\_INT procedure returns with this bit set, another SI interrupt will occur.

The SIU\_INT procedure branches three independent cases. The first case is entered if the STATION\_STATE is not in the I\_T\_S. If this is true, then the SIU is not in the AUTO mode, and the CPU will have to respond to the primary on its own. (Remember that the AUTO mode is entered when the STATION\_STATE enters into I\_T\_S.) If the STATION\_STATE is in the I\_T\_S, then either the SIU has just left the AUTO mode, or is still in the AUTO mode. This is the second and third case respectively.

In the first case, if the STATION\_STATE is not in the I\_T\_S, then it must be in either the L\_D\_S or the FRMR\_S. In either case a separate procedure is called based on which state the station is in. The In\_Disconnect\_State procedure sends to the primary a DM response, unless it received a SNRM command and the USER\_STATE equals open. In that case the SIU sends an UA and enters into the I\_T\_S. The In\_FRMR\_State procedure will send the primary the FRMR response unless it received either a DISC or an SNRM. If the primary's command was a DISC, then the secondary will send an UA and enter into the L\_D\_S. If the primary's command was a SNRM, then the secondary will send an UA, enter into the I\_T\_S, and clear NSNR register.

For the second case, if the STATION\_STATE is in the I\_T\_S but the SIU left the AUTO mode, then the CPU must determine why the AUTO mode was exited, and generate a response to the primary. There are four reasons for the SIU to automatically leave the AUTO mode. The following is a list of these reasons, and the responses given by the SSD based on each reason.

1. The SIU has received a command field it does not recognize.

Response: If the CPU recognizes the command, it generates the appropriate response. If neither the SIU nor the CPU recognize the command, then a FRMR response is sent.

2. The SIU has received a Sequence Error Sent (SES=1 in NSNR register).  $Nr(P) \neq Ns(S)+1$ , and  $Nr(P) \neq Ns(S)$ .

Response: Send FRMR.

3. A buffer overrun has occurred. BOV=1 in STS register.

Response: Send FRMR.

4. An I frame with data was received while RPB=1.

Response: Go back into AUTO mode and send an AUTO mode response.

In addition to the above reasons, there is one condition where the CPU forces the SIU out of the AUTO mode. This is discussed in the SSD's User Interface Procedures section in the CLOSED procedure description.

Finally, case three is when the STATION\_STATE is in the I\_T\_S and the AUTO mode. The CPU first looks at the TBF bit. If this bit is 0 then the interrupt may have been caused by a frame which was transmitted and acknowledged. Therefore the XMIT\_BUFFER\_EMPTY flag is set again indicating that the application software can transmit another frame.

The other reason this section of code could be entered is if a valid I frame was received. When a good I frame is received the RBE bit equals 0. This means that the receiver is disabled. If the primary were to poll the 8044 while RBE=0, it would time out since no response would be given. Time outs reduce network throughput. To improve network performance, the CPU first sets RBP, then sets RBE. Now when the primary polls the 8044 an immediate RNR response is given. At this point the SSD calls the application software procedure SIU\_RECV and passes the length of the data as a parameter. The SIU\_RECV procedure reads the data out of the receive buffer then returns to the SSD module. Now that the receive information has been transferred, RBP can be cleared.

### Command\_Decode Procedure

The Command\_Decode procedure is called from the SIU\_INT procedure when the STATION\_STATE = I\_T\_S and the SIU left the AUTO mode as a result

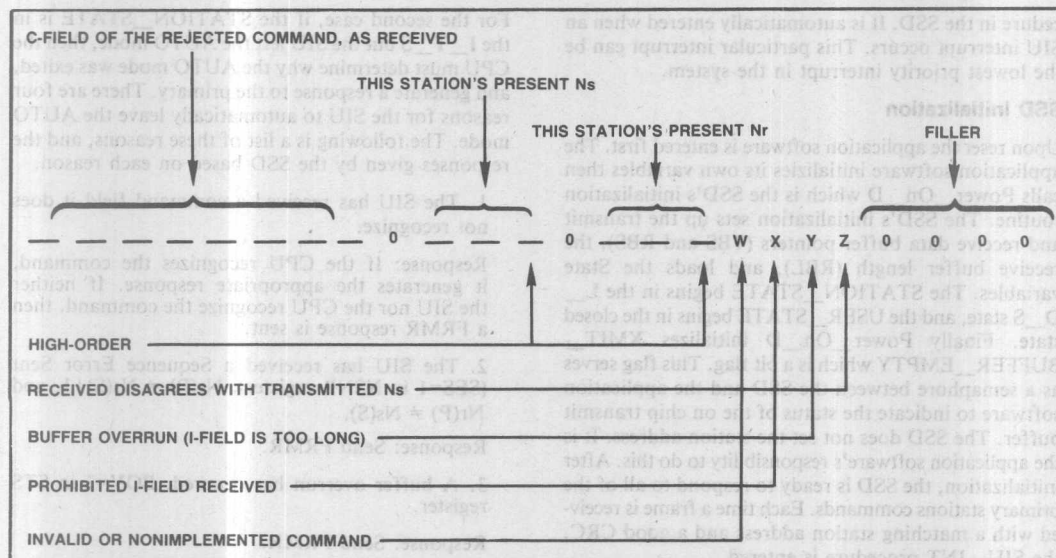


Figure 19-11. Information Field of the FRMR Response, as Transmitted

of not being able to recognize the receive control byte. Commands which the SIU AUTO mode does not recognize are handled here. The commands recognized in this procedure are: SNRM, DISC, and TEST. Any other command received will generate a Frame Reject with the nonimplemented command bit set in the third data byte of the FRMR frame. Any additional unnumbered frame commands which the secondary station is going to implement, should be implemented in this procedure.

If a SNRM is received the command\_decode procedure calls the SNRM\_Response procedure. The SNRM\_Response procedure sets the STATION\_STATE = I\_T\_S, clears the NSNR register and responds with an UA frame. If a DISC is received, the command\_decode procedure sets the STATION\_STATE = L\_D\_S, and responds with an UA frame. When a TEST frame is received, and there is no buffer overrun, the command\_decode procedure responds with a TEST frame retransmitting the same data it received. However if a TEST frame is received and there is a buffer overrun, then a TEST frame will be sent without any data, instead of a FRMR with the buffer overrun bit set.

#### Frame Reject Procedures

There are two procedures which handle the FRMR state: XMIT\_FRMR and IN\_FRMR\_STATE. XMIT\_FRMR is entered when the secondary station first goes into the FRMR state. The frame reject response frame contains the FRMR response in the command field plus three additional data bytes in the I field. Figure 19-11 displays the format for the three data byte in the I field

of a FRMR response. The XMIT\_FRMR procedure sets up the Frame Reject response frame based on the parameter REASON which is passed to it. Each place in the SSD code that calls the XMIT\_FRMR procedure, passes the REASON that this procedure was called, which in turn is communicated to the primary station. The XMIT\_FRMR procedure uses three bytes of internal RAM which it initializes for the correct response. The TBS and TBL registers are then changed to point to the FRMR buffer so that when a response is sent these three bytes will be included in the I field,

The IN\_FRMR\_STATE procedure is called by the SIU\_INT procedure when the STATION\_STATE already is in the FRMR state and a response is required. The IN\_FRMR\_STATE procedure will only allow two commands to remove the secondary station from the FRMR state: SNRM and DISC. Any other command which is received while in the FRMR state will result a FRMR response frame.

#### XMIT\_UNNUMBERED Procedure

This is a general purpose transmit procedure, used only in the FLEXIBLE mode, which sends unnumbered responses to the primary. It accepts the control byte as a parameter, and also expects the TBL register to be set before the procedure is called. This procedure waits until the frame has been transmitted before returning. If this procedure returned before the transmit interrupt was generated, the SIU\_INT routine would be entered. The SIU\_INT routine would not be able to distinguish this condition.

SSD's User Interface Procedures -- OPEN, CLOSE, TRANSMIT, SIU\_RECV -- are discussed in the following section.

The OPEN procedure is the simplest of all, it changes the USER\_STATE to OPEN\_S then returns. This lets the SSD know that the user wants to open the channel for communications. When the SSD receives a SNRM command, it checks the USER\_STATE. If the USER\_STATE is open, then the SSD will respond with an UA, and the STATION\_STATE enters the I\_T\_S.

The CLOSE procedure is also simple, it changes the USER\_STATE to CLOSED\_S and sets the AM bit to 0. Note that when the CPU sets the AM bit to 0 it puts the SIU out of the AUTO mode. This event is asynchronous to the events on the network. As a result an I frame can be lost. This is what can happen.

1. AM is set to 0 by the CLOSE Procedure.
2. An I frame is received and a SI interrupt occurs.
3. The SIU\_INT procedure enters case 2. (STATION\_STATE = I\_T\_S, and AM = 0)
4. Case 2 detects that the USER\_STATE = CLOSED\_S, sends a RD response and ignores the fact that an I frame was received.

Therefore it is advised to never call the CLOSE procedure or take the SIU out of the AUTO mode when it is receiving I frames or an I frame will be lost.

For both the TRANSMIT and SIU\_RECV procedures, it is the application software's job to put data into the transmit buffer, and take data out of the receive buffer. The SSD does not transfer data in or out of its transmit or receive buffers because it does not know what kind of buffering the application software is implementing. What the SSD does do is notify the application software when the transmit buffer is empty, XMIT\_BUFFER\_EMPTY = 1, and when the receive buffer is full.

One of the functions that the SSD performs to synchronize the application software to the SDLC data link. However some of the synchronization must also be done by the application software. Remember that the SSD does not want to hang up the application software waiting for some event to occur on the SDLC data link, therefore the SSD always returns to the application software as soon as possible.

For example, when the application software calls the OPEN procedure, the SSD returns immediately. The application software thinks that the SDLC channel is now open and it can transmit. This is not the case. For the channel to be open, the SSD must receive a SNRM from the primary and respond with a UA. However, the SSD does not want to hang up the application software waiting for a SNRM from the primary before returning from the OPEN procedure. When the

TRANSMIT procedure is called, the SSD expects the STATION\_STATE to be in the I\_T\_S. If it isn't, the SSD refuses to transmit the data. The TRANSMIT procedure first checks to see if the USER\_STATE is open, if not the USER\_STATE\_CLOSED parameter is passed back to the application module. The next thing TRANSMIT checks is the STATION\_STATE. If this is not open, then TRANSMIT passes back LINK\_DISCONNECTED. This means that the USER\_STATE is open, but the SSD hasn't received a SNRM command from the primary yet. Therefore, the application software should wait awhile and try again. Based on network performance, one knows the maximum amount of time it will take for a station to be polled. If the application software waits this length of time and tries again but still gets a LINK\_DISCONNECTED parameter passed back, higher level recovery must be implemented.

Before loading the transmit buffer and calling the TRANSMIT procedure, the application software must check to see that XMIT\_BUFFER\_EMPTY = 1. This flag tells the application software that it can write new data into the transmit buffer and call the TRANSMIT procedure. After the application software has verified that XMIT\_BUFFER\_EMPTY = 1, it fills the transmit buffer with the data and calls the TRANSMIT procedure passing the length of the buffer as a parameter. The TRANSMIT procedure checks for three reasons why it might not be able to transmit the frame. If any of these three reasons are true, the TRANSMIT procedure returns a parameter explaining why it couldn't send the frame. If the application software receives one of these responses, it must rectify the problem and try again. Assuming these three conditions are false, then the SSD clears XMIT\_BUFFER\_EMPTY, attempts to send the data and returns the parameter DATA\_TRANSMITTED. XMIT\_BUFFER\_EMPTY will not be set to 1 again until the data has been transmitted and acknowledged.

The SIU\_RECV procedure must be incorporated into the application software module. When a valid I frame is received by the SIU, it calls the SIU\_RECV procedure and passes the length of the received data as a parameter. The SIU\_RECV procedure must remove all of the data from the receive buffer before returning to the SIU\_INT procedure.

### Linking up to the SSD

Figure 19-12 shows necessary parts to include in a PL/M-51 application program that will be linked to the SSD module. RL51 is used to link and locate the SSD and application modules. The command line used to do this is:



RL51.SSD.obj,filename.obj,PLM51.LIB TO filename  
& RAMSIZE(192)

```
$registerbank(0)
user$mod: do;
$include (reg44.dcl)
declare
  lit          literally 'literally',
  -buffer__length  lit          '60',
  siu__xmit__buffer  byte      external idata,
  (buffer__length)  byte      external,
  siu__rcv__buffer  bit        external;
  (buffer__length)  bit        external;
  xmit__buffer__empty  bit        external;
/* external procedures */

power__on__d: procedure      external;
end power__on__d;

close: procedure            external      using 1;
end close;

open: procedure             external      using 1;
end open;

transmit: procedure
(xmit__buffer__length)  byte      external;
  declare  xmit__buffer__length  byte;
end transmit;

/* local procedures */

siu__rcv: procedure (length) public      using 1;
  declare  length  byte;
end siu__rcv;
```

**Figure 19-12. Applications Module Link Information**

#### PL/M-51 and Register Banks

The 8044 has four register banks. PL/M-51 assumes that an interrupt procedure never uses the same bank as the procedure it interrupts. The USING attribute of a procedure, or the \$REGISTERBANK control, can be used to ensure that.

The SSD module uses the \$REGISTERBANK(1) attribute. Some procedures are modified with the USING attribute based on the register bank level of the calling procedure.

#### 19.2.5 APPLICATION MODULE; Async to SDLC protocol converter

One of the purposes of this application module is to demonstrate how to interface software to the SSD. Another purpose is to implement and test a practical application. This application software performs I/O with an async terminal through a USART, buffers data, and also performs I/O with the SSD. In addition, it allows the user on the async terminal to: set the station ad-

dress, set the destination address, and go online and offline. Setting the station address sets the byte in the STAD register. The destination address is the first byte in the I field. Going online or offline results in either calling the OPEN or CLOSE procedure respectively.

After the secondary station powers up, it enters the 'terminal mode', which accepts data from the terminal. However, before any data is sent, the user must configure the station. The station address and destination address must be set, and the station must be placed online. To configure the station the ESC character is entered at the terminal which puts the protocol converter into the 'configure mode'. Figure 20-13 shows the menu which appears on the terminal screen.

#### ( / ) 8044 Secondary Station

- 1 - Set the Station Address
- 2 - Set the Destination Address
- 3 - Go Online
- 4 - Go Offline
- 5 - Return to terminal mode

Enter option —

**Figure 19-13. Menu for the Protocol Converter**

In the terminal mode data is buffered up in the secondary station. A Line Feed character 'LF' tells the secondary station to send an I frame. If more than 60 bytes are buffered in the secondary station when a 'LF' is received, the applications software packetizes the data into 60 bytes or less per frame. If a LF is entered when the station is offline, an error message comes on the screen which says 'Unable to Get Online'.

The secondary station also does error checking on the async interface for Parity, Framing Error, and Over-run Error. If one of these errors are detected, an error message is displayed on the terminal screen.

#### Buffering

There are two separate buffers in the application module: a transmit buffer and a receive buffer. The transmit buffer receives data from the USART, and sends data to the SSD. The receive buffer receives data from the SSD, and transmits data to the USART. Each buffer is a 256 byte software FIFO. If the transmit FIFO becomes full and no 'LF' character is received, the secondary station automatically begins sending the data. In addition, the application module will shut off the terminal's transmitter using CTS until the FIFO has been partially emptied. A block diagram of the buffering for the protocol converter is given in Figure 19-14.

#### Application Module Software

A block diagram of the application module software is given in Figure 19-15. There are three interrupt



routines in this module: `USART_RECV_INT`, `USART_XMIT_INT`, and `TIMER_0_INT`. The first two are for servicing the USART. `TIMER_0_INT` is used if the TRANSMIT procedure in the SSD is called and does not return with the `DATA_TRANSMITTED` parameter. `TIMER_0_INT` employs Timer 0 to wait a finite amount of time before trying to transmit again. The highest priority interrupt is `USART_RECV_INT`. The main program and all the procedures it calls use register bank 0, `USART_XMIT_INT` and `TIMER_0_INT` and `FIFO_R_OUT` use bank 1, while `USART_RECV_INT` and all the procedures it calls use register bank 2.

### Power\_On Procedure

The `Power_On` procedure initializes all of the chips in the system including the 8044. The 8044 is initialized to use the on-chip DPLL with NRZI coding, PreFrame Sync, and Timer 1 auto reload at a baud rate of 62.5 Kbps. The 8254 and the 8251A are initialized next based on the DIP switch values attached to port 1 on the 8044. Variables and pointers are initialized, then the SSD's Power-Up Procedure, `Power_On`, is called. Finally the interrupt system is enabled and the main program is entered.

### Main Program

The main program is a simple loop which waits for a frame transmit command. A frame transmit command is indicated when the variable `SEND_DATA` is greater than 0. The value of `SEND_DATA` equals the number of 'LF' characters in the transmit FIFO, hence it also indicates the number of frames pending transmission. Each time a frame is sent, `SEND_DATA` is decremented by one. Thus when `SEND_DATA` is greater than 0, the main program falls down into the next loop which polls the `XMIT_BUFFER_EMPTY` bit. When `XMIT_BUFFER_EMPTY` equals 1, the `SIU_XMIT_BUFFER` can be loaded. The first byte in the buffer is loaded with the destination address while the rest of the buffer is loaded with the data. Bytes are removed from the transmit FIFO and placed into the `SIU_XMIT_BUFFER` until one of three things happen: 1. a 'LF' character is read out of the FIFO, 2. the number of bytes loaded equals the size of the `SIU_XMIT_BUFFER`, or 3. the transmit FIFO is empty.

After the `SIU_XMIT_BUFFER` is filled, the SSD TRANSMIT procedure is called and the results from the procedure are checked. Any result other than `DATA_TRANSMITTED` will result in several retries within a finite amount of time. If all the retries fail then the `LINK_DISC` procedure is called which sends a message to the terminal, 'Unable to Get Online'.

### USART\_RECV\_INT Procedure

When the 8251A receives a character, the `RxRDY` pin

on the 8251A is activated, and this interrupt procedure is entered. The routine reads the USART status register to determine if there are any errors in the character received. If there are, the character is discarded and the `ERROR` procedure is called which prints the type of error on the screen. If there are no errors, the received character is checked to see if it's an ESC. If it is an ESC, the `MENU` procedure is called which allows the user to change the configuration. If neither one of these two conditions exits the received character is inserted into the transmit FIFO. The received character may or may not be echoed back to the terminal based on the dip switch settings.

### Transmit FIFO

The transmit FIFO consists of two procedures: `FIFO_T_IN` and `FIFO_T_OUT`. `FIFO_T_IN` inserts a character into the FIFO, and `FIFO_T_OUT` removes a character from the FIFO. The FIFO itself is an array of 256 bytes called `FIFO_T`. There are two pointers used as indexes in the array to address the characters: `IN_PTR_T` and `OUT_PTR_T`. `IN_PTR_T` points to the location in the array which will store the next byte of data inserted. `OUT_PTR_T` points to the next byte of data removed from the array. Both `IN_PTR_T` and `OUT_PTR_T` are declared as bytes. The `FIFO_T_IN` procedure receives a character from the `USART_RECV_INT` procedure and stores it in the array location pointed to by `IN_PTR_T`, then `IN_PTR_T` is incremented. Similarly, when `FIFO_T_OUT` is called by the main program, to load the `SIU_XMIT_BUFFER`, the byte in the array pointed to by `OUT_PTR_T` is read, then `OUT_PTR_T` is incremented. Since `IN_PTR_T` and `OUT_PTR_T` are always incremented, they must be able to roll over when they hit the top of the 256 byte address space. This is done automatically by having both `IN_PTR_T` and `OUT_PTR_T` declared as bytes. Each character inserted into the transmit FIFO is tested to see if it's a LF. If it is a LF, the variable `SEND_DATA` is incremented which lets the main program know that it is time to send an I frame. Similarly each character removed from the FIFO is tested. `SEND_DATA` is decremented for every LF character removed from the FIFO.

`IN_PTR_T` and `OUT_PTR_T` are also used to indicate how many bytes are in the FIFO, and whether it is full or empty. When a character is placed into the FIFO and `IN_PTR_T` is incremented, the FIFO is full if `IN_PTR_T` equals `OUT_PTR_T`. When a character is read from the FIFO and `OUT_PTR_T` is incremented, the FIFO is empty if `OUT_PTR_T` equals `IN_PTR_T`. If the FIFO is neither full nor empty, then it is in use. A byte called `BUFFER_STATUS_T` is used to indicate one of these three conditions. The application module uses the buffer status information to control the flow of data into and out of the FIFO. When the transmit FIFO is empty, the main program must stop loading bytes into the `SIU_`

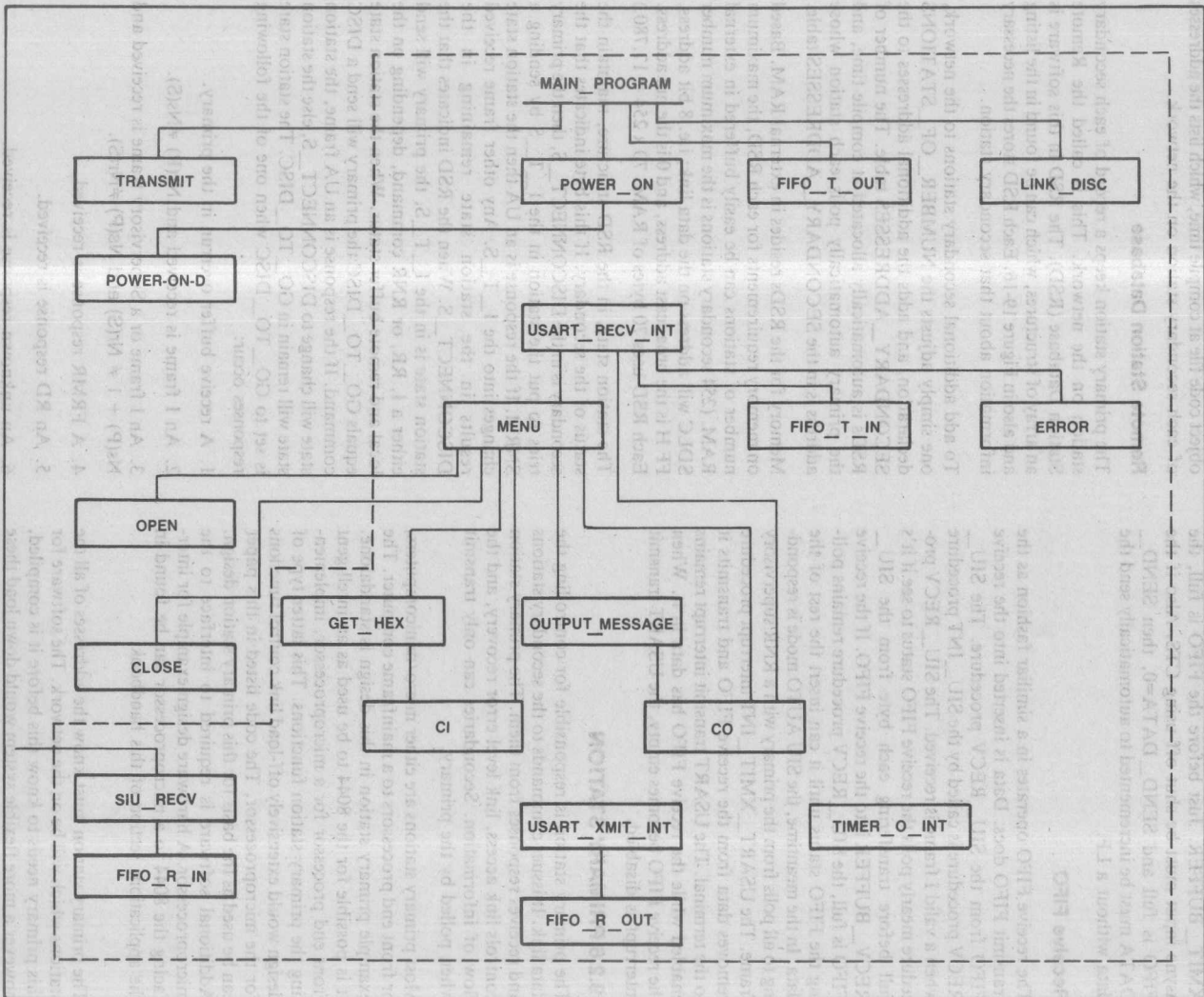


Figure 19-15. Block Diagram of User Software



**XMIT\_BUFFER.** Just before the FIFO is full, the async input must be shut off using CTS. Also if the FIFO is full and **SEND\_DATA=0**, then **SEND\_DATA** must be incremented to automatically send the data without a LF.

### Receive FIFO

The receive FIFO operates in a similar fashion as the transmit FIFO does. Data is inserted into the receive FIFO from the **SIU\_RECV** procedure. The **SIU\_RECV** procedure is called by the **SIU\_INT** procedure when a valid I frame is received. The **SIU\_RECV** procedure nearly polls the receive FIFO status to see if it's full before transferring each byte from the **SIU\_RECV\_BUFFER** into the receive FIFO. If the receive FIFO is full, the **SIU\_RECV** procedure remains polling the FIFO status until it can insert the rest of the data. In the meantime, the **SIU\_AUTO** mode is responding to all polls from the primary with a RNR supervisory frame. The **USART\_XMIT\_INT** interrupt procedure removes data from the receive FIFO and transmits it to the terminal. The **USART** transmit interrupt remains enabled while the receive FIFO has data in it. When the receive FIFO becomes empty, the **USART** transmit interrupt is disabled.

### 19.2.6 PRIMARY STATION

The primary station is responsible for controlling the data link. It issues commands to the secondary stations and receives responses from them. The primary station controls link access, link level error recovery, and the flow of information. Secondaries can only transmit when polled by the primary.

Most primary stations are either micro/minicomputers, or front end processors to a mainframe computer. The example primary station in this design is standalone. It is possible for the 8044 to be used as an intelligent front end processor for a microprocessor, implementing the primary station functions. This latter type of design would extensively off-load link control functions for the microprocessor. The code listed in this paper can be used as the basis for this primary station design. Additional software is required to interface to the microprocessor. A hardware design example for interfacing the 8044 to a microprocessor can be found in the applications section of this handbook.

The primary station must know the addresses of all the stations which will be on the network. The software for this primary needs to know this before it is compiled, however a more flexible system would download these parameters.

From the listing of the software it can be seen that the variable **NUMBER\_OF\_STATIONS** is a literal declaration, which is 2 in this design example. There were three stations tested on this data link, two secondaries and one primary. Following the **NUMBER\_OF\_STATIONS** declaration is a table, loaded into the

object code file at compile time, which lists the addresses of each secondary station on the network.

### Remote Station Database

The primary station keeps a record of each secondary station on the network. This is called the Remote Station Database (RSD). The RSD in this software is an array of structures, which can be found in the listing and also in Figure 19-16. Each RSD stores the necessary information about that secondary station.

To add additional secondary stations to the network, one simply adjusts the **NUMBER\_OF\_STATIONS** declaration, and adds the additional addresses to the **SECONDARY\_ADDRESSES** table. The number of RSDs is automatically allocated at compile time, and the primary automatically polls each station whose address is in the **SECONDARY\_ADDRESSES** table.

Memory for the RSDs resides in external RAM. Based on memory requirements for each RSD, the maximum number of stations can be easily buffered in external RAM. (254 secondary stations is the maximum number SDLC will address on the data link; i.e. 8 bit address, FF H is the broadcast address, and 0 is the nul address. Each RSD uses 70 bytes of RAM.  $70 \times 254 = 17,780$ .)

The station state, in the RSD structure, maintain the status of the secondary. If this byte indicates that the secondary is in the **DISCONNECT\_S**, then the primary tries to put the station in the **I\_T\_S** by sending a SNRM. If the response is an UA then the station state changes into the **I\_T\_S**. Any other frame received results in the station state remaining in the **DISCONNECT\_S**. When the RSD indicates that the station state is in the **I\_T\_S**, the primary will send either a I, RR, or RNR command, depending on the local and remote buffer status. When the station state equals **GO\_TO\_DISC** the primary will send a DISC command. If the response is an UA frame, the station state will change to **DISCONNECT\_S**, else the station state will remain in **GO\_TO\_DISC**. The station state is set to **GO\_TO\_DISC** when one of the following responses occur:

1. A receive buffer overrun in the primary.
2. An I frame is received and  $Nr(P) \neq Ns(S)$ .
3. An I frame or a Supervisory frame is received and  $Ns(P) + 1 \neq Nr(S)$  and  $Ns(P) \neq Nr(S)$ .
4. A FRMR response is received.
5. An RD response is received.
6. An unknown response is received.

The send count (Ns) and receive count (Nr) are also maintained in the RSD. Each time an I frame is sent by the primary and acknowledged by the secondary, Ns is incremented. Nr is incremented each time a valid I frame is received. **BUFFER\_STATUS** indicates the status of the secondary stations buffer. If a RR response is received, **BUFFER\_STATUS** is set to **BUFFER\_**

READY. If a RNR response is received, BUFFER\_STATUS is set to BUFFER\_NOT\_READY.

### Buffering

The buffering for the primary station is as follows: within each RSD is a 64 byte array buffer which is initially empty. When the primary receives an I frame, it looks for a match between the first byte of the I frame and the addresses of the secondaries on the network. If a match exists, the primary places the data in the RSD buffer of the destination station. The INFO\_LENGTH in the RSD indicates how many bytes are in the buffer. If INFO\_LENGTH equals 0, then the buffer is empty. The primary can buffer only one I frame per station. If a second I frame is received while the addressed secondary's RSD buffer is full, the primary cannot receive any more I frames. At this point the primary continues to poll the secondaries using RNR supervisory frame.

### Primary Station Software

A block diagram of the primary station software is shown in Figure 19-17. The primary station software consists of a main program, one interrupt routine, and

several procedures. The POWER\_ON procedure begins by initializing the SIU's DMA and enabling the receiver. Then each RSD is initialized. The DPLL and the timers are set, and finally the TIMER 0 interrupt is enabled.

The main program consists of an iterative do loop within a do forever loop. The iterative do loop polls each secondary station once through the do loop. The variable STATION\_NUMBER is the counter for the iterative do statement which is also used as an index to the array of RSD structures. The primary station issues one command and receives one response from every secondary station each time through the loop. The first statement in the loop loads the secondary station address, indexed by STATION\_NUMBER into the array of the RSD structures. Now when the primary sends a command, it will have the secondary's address in the address field of the frame. The automatic address recognition feature is used by the primary to recognize the response from the secondary.

Next the main program determines the secondary stations state. Based on this state, the primary knows what command to send. If the station is in the DISCONNECT\_S, the primary calls the SNRM\_P

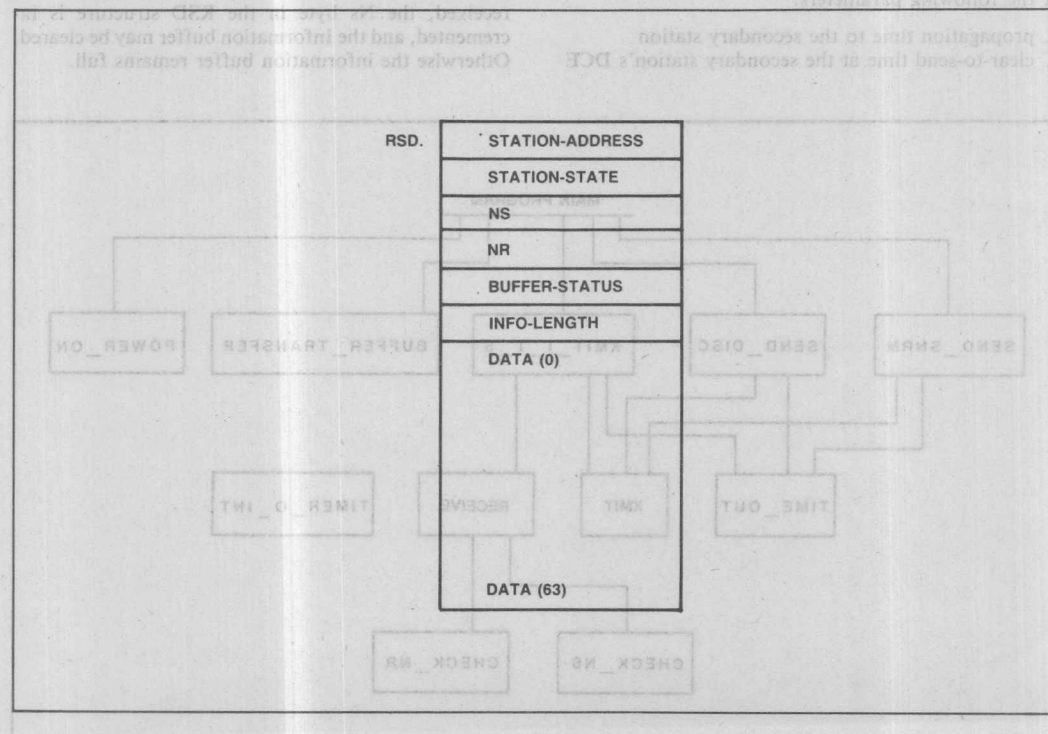


Figure 19-16. Remote Station Database Structure

procedure to try and put the secondary in the I\_T\_S. If the station state is in the GO\_TO\_DISC state, the DISC\_P is called to try and put the secondary in the L\_D\_S. If the secondary is in neither one of the above two states, then it is in the I\_T\_S. When the secondary is in the I\_T\_S, the primary could send one of three commands: I, RR, or RNR. If the RSD's buffer has data in it, indicated by INFO\_LENGTH being greater than zero, and the secondary's BUFFER\_STATUS equals BUFFER\_READY, then an I frame will be sent. Else if RPB=0, a RR supervisory frame will be sent. If neither one of these cases is true, then an RNR will be sent. The last statement in the main program checks the RPB bit. If set to one, the BUFFER\_TRANSFER procedure is called, which transfers the data from the SIU receive buffer to the appropriate RSD buffer.

### Receive Time Out

Each time a frame is transmitted, the primary sets a receive time out timer; Timer 0. If a response is not received within a certain time, the primary returns to the main program and continues polling the rest of the stations. The minimum length of time the primary should wait for a response can be calculated as the sum of the following parameters.

1. propagation time to the secondary station
2. clear-to-send time at the secondary station's DCE

3. appropriate time for secondary station processing
4. propagation time from the secondary station
5. maximum frame length time

The clear-to-send time and the propagation time are negligible for a local network at low bit rates. However, the turnaround time and the maximum frame length time are significant factors. Using the 8044 secondaries in the AUTO mode minimizes turnaround time. The maximum frame length time comes from the fact the 8044 does not generate an interrupt from a received frame until it has been completely received, and the CRC is verified as correct. This means that the time-out is bit rate dependent.

### Ns and Nr check Procedures

Each time an I frame or supervisory frame is received, the Nr field in the control byte must be checked. Since this data link only allows one outstanding frame, a valid Nr would satisfy either one of two equations;  $Ns(P) + 1 = Nr(S)$  the I frame previously sent by the primary is acknowledged,  $Ns(P) = Nr(S)$  the I frame previously sent is not acknowledged. If either one of these two cases is true, the CHECK\_NR procedure returns a parameter of TRUE; otherwise a FALSE parameter is returned. If an acknowledgement is received, the Ns byte in the RSD structure is incremented, and the Information buffer may be cleared. Otherwise the information buffer remains full.

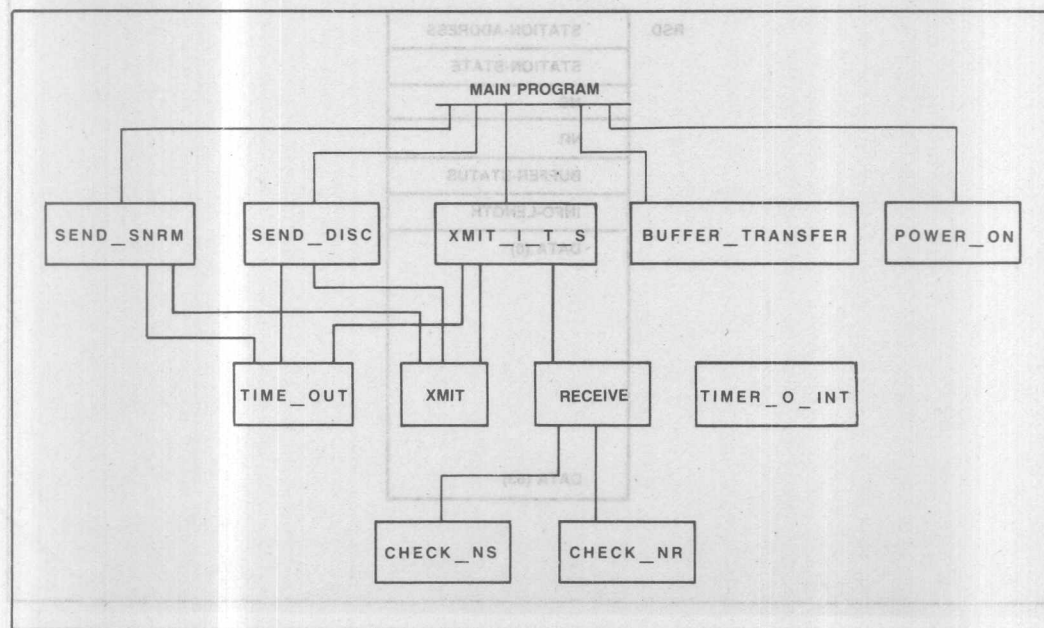


Figure 19-17. Block Diagram of Primary Station Software Structure

When an I frame is received, the Ns field has to be checked also. If  $Nr(P) = Ns(S)$ , then the procedure returns TRUE, otherwise a FALSE is returned.

### Receive Procedure

The receive procedure is called when a supervisory or information frame is sent, and a response is received before the time-out period. The RECEIVE procedure can be broken down into three parts. The first part is entered if an I frame is received. When an I frame is received, Ns, Nr and buffer overrun are checked. If there is a buffer overrun, or there is an error in either Ns or Nr, then the station state is set to GO\_TO\_DISC. Otherwise Nr in the RSD is incremented, the receive field length is saved, and the RPB bit is set. By incrementing the Nr field, the I frame just received is acknowledged the next time the primary polls the secondary with an I frame or a supervisory frame. Setting RBP protects the received data, and also tells

the main program that there is data to transfer to one of the RSD buffers.

If a supervisory frame is received, the Nr field is checked. If a FALSE is returned, then the station state is set to GO\_TO\_DISC. If the supervisory frame received was an RNR, buffer status is set to not ready. If the response is not an I frame, nor a supervisory frame, then it must be an Unnumbered frame.

The only Unnumbered frames the primary recognizes are UA, DM, and FRMR. In any event, the station state is set to GO\_TO\_DISC. However if the frame received is a FRMR, Nr in the second data byte of the I field is checked to see if the secondary acknowledged an I frame received before it went into the FRMR state. If this is not done and the secondary acknowledged an I frame which the primary did not recognize, the primary transmits, the I frame when the secondary returns to the I\_T\_S. In this case, the secondary would receive duplicate I frames.



The main program that there is data to transfer to one of the RSD buffers.

If a supervisory frame is received, the Nr field is checked. If a FALSE is returned, then the station state is set to GO\_TO\_DISC. If the supervisory frame is received was an RNR, buffer status is set to not ready. If the response is not an I frame, not a supervisory frame, then it must be an Unnumbered frame.

The only Unnumbered frames the primary recognizes are UA, DM, and FRMR. In any event, the station state is set to GO\_TO\_DISC. However, if the frame received is a FRMR, Nr in the second data byte of the I field is checked to see if the secondary acknowledged it. If it is done and the secondary acknowledged it, then the primary will not recognize the frame. If the primary transmits the I frame when the secondary returns to the T\_S. In this case, the secondary would receive duplicate I frames.

When an I frame is received, the Nr field has to be checked also. If  $Nr(P) = Nr(S)$ , then the procedure returns TRUE, otherwise a FALSE is returned.

# Receive Procedure

The receive procedure is called when a supervisory or information frame is sent, and a response is received before the time-out period. The RECEIVE procedure can be broken down into three parts. The first part is to check if an I frame is received. When an I frame is received, Nr and buffer overrun are checked. If there is a buffer overrun, or there is an error in either Nr or Nr, then the station state is set to GO\_TO\_DISC. Otherwise Nr in the RSD is incremented, the receive field length is saved, and the RPB bit is incremented the Nr field. The primary polls the secondary the next time the primary polls the secondary with an I frame or a supervisory frame. Setting RBP protects the received data, and also tells

## APPENDIX A 8044 SOFTWARE FLOWCHARTS

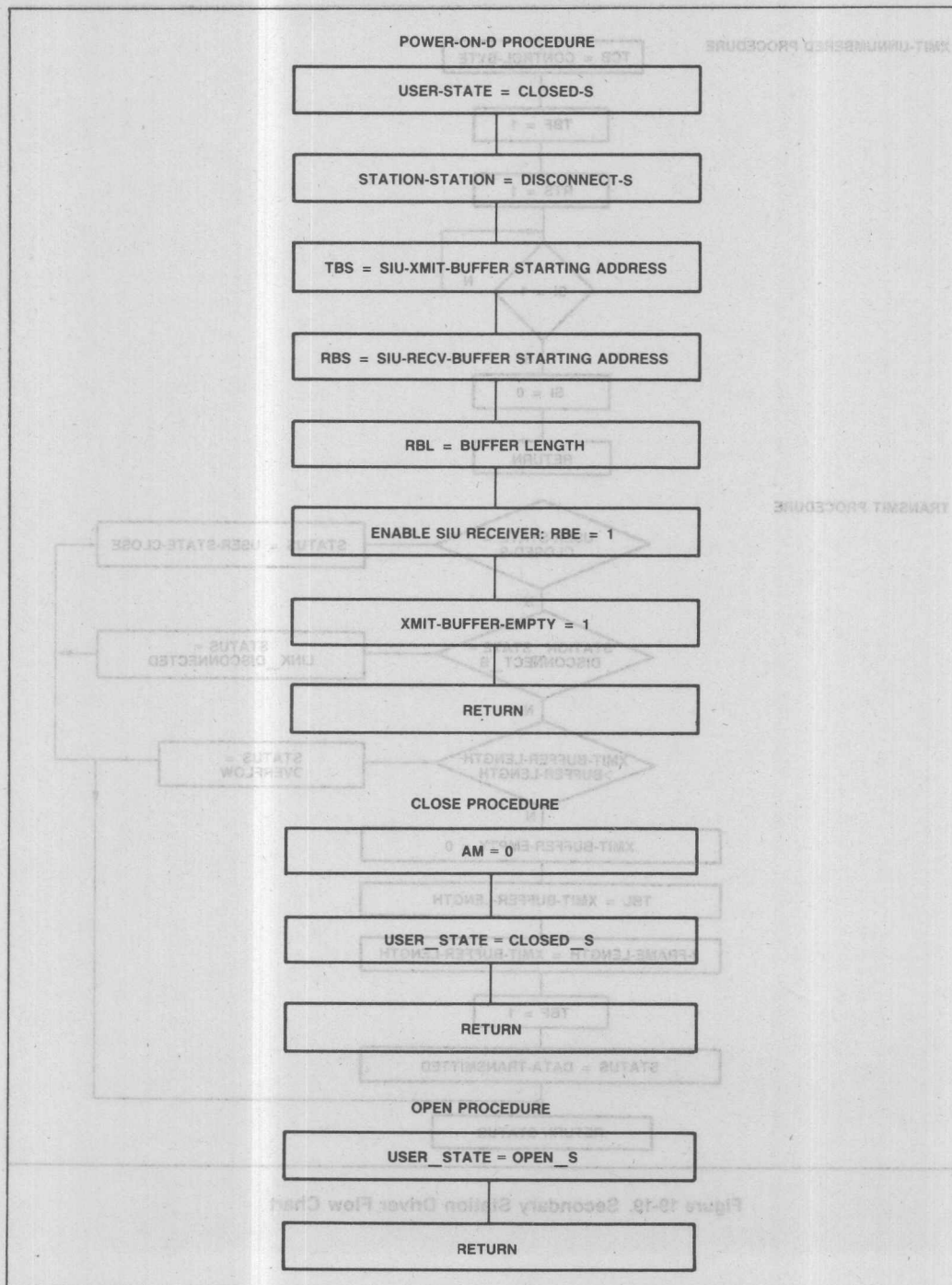


Figure 19-18. Secondary Station Driver Flow Chart

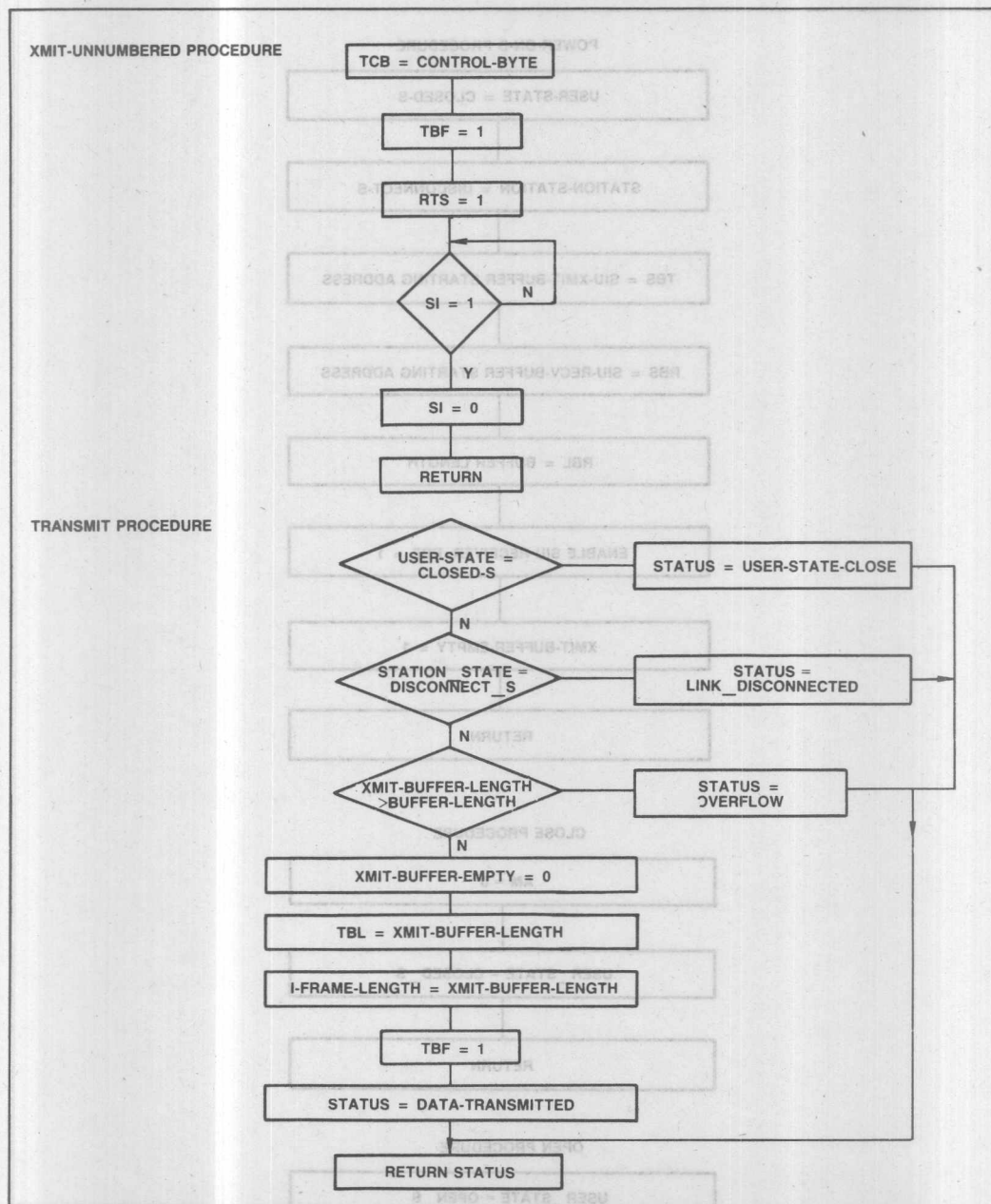


Figure 19-19. Secondary Station Driver Flow Chart

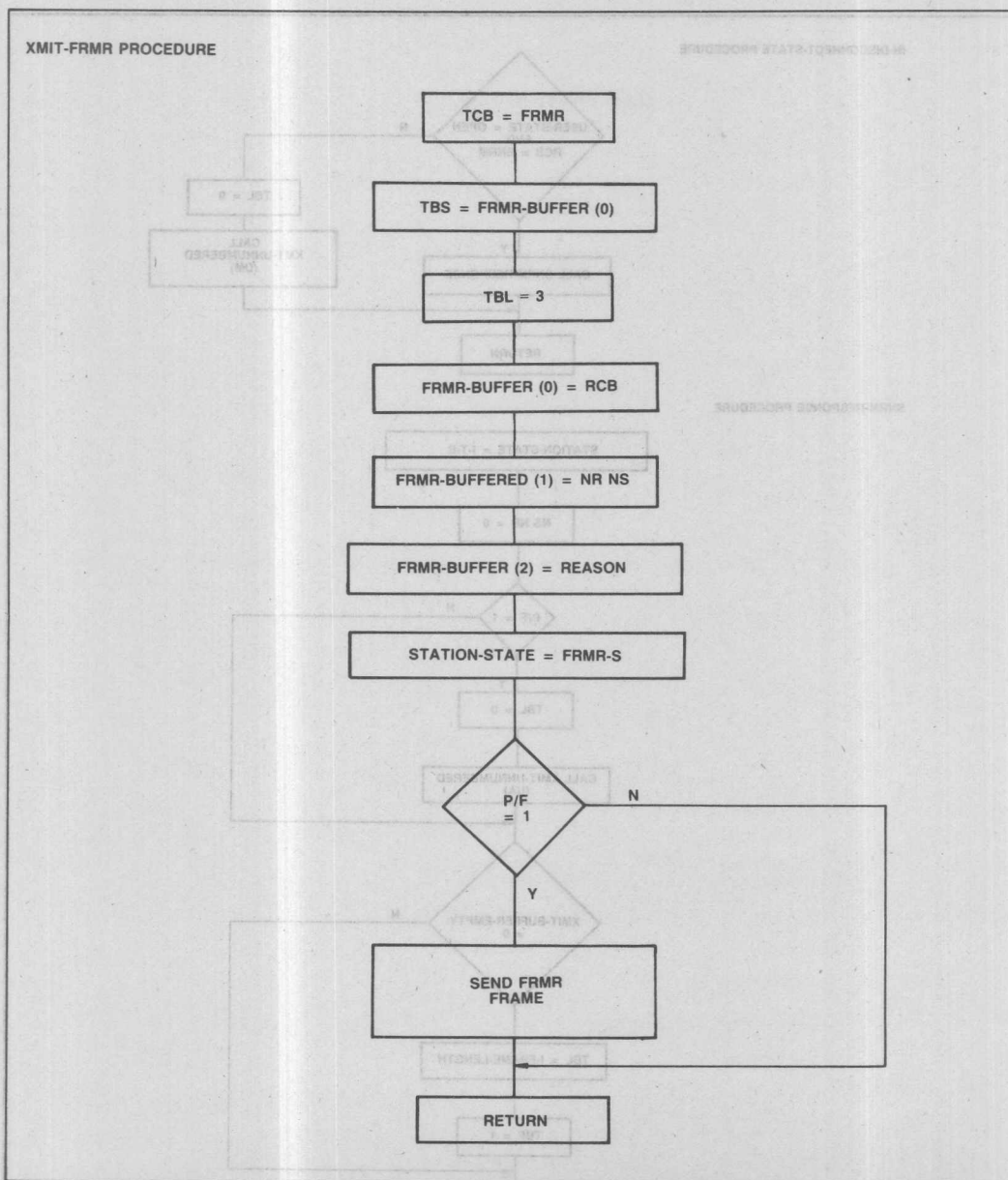


Figure 19-20. Secondary Station Driver Flow Chart



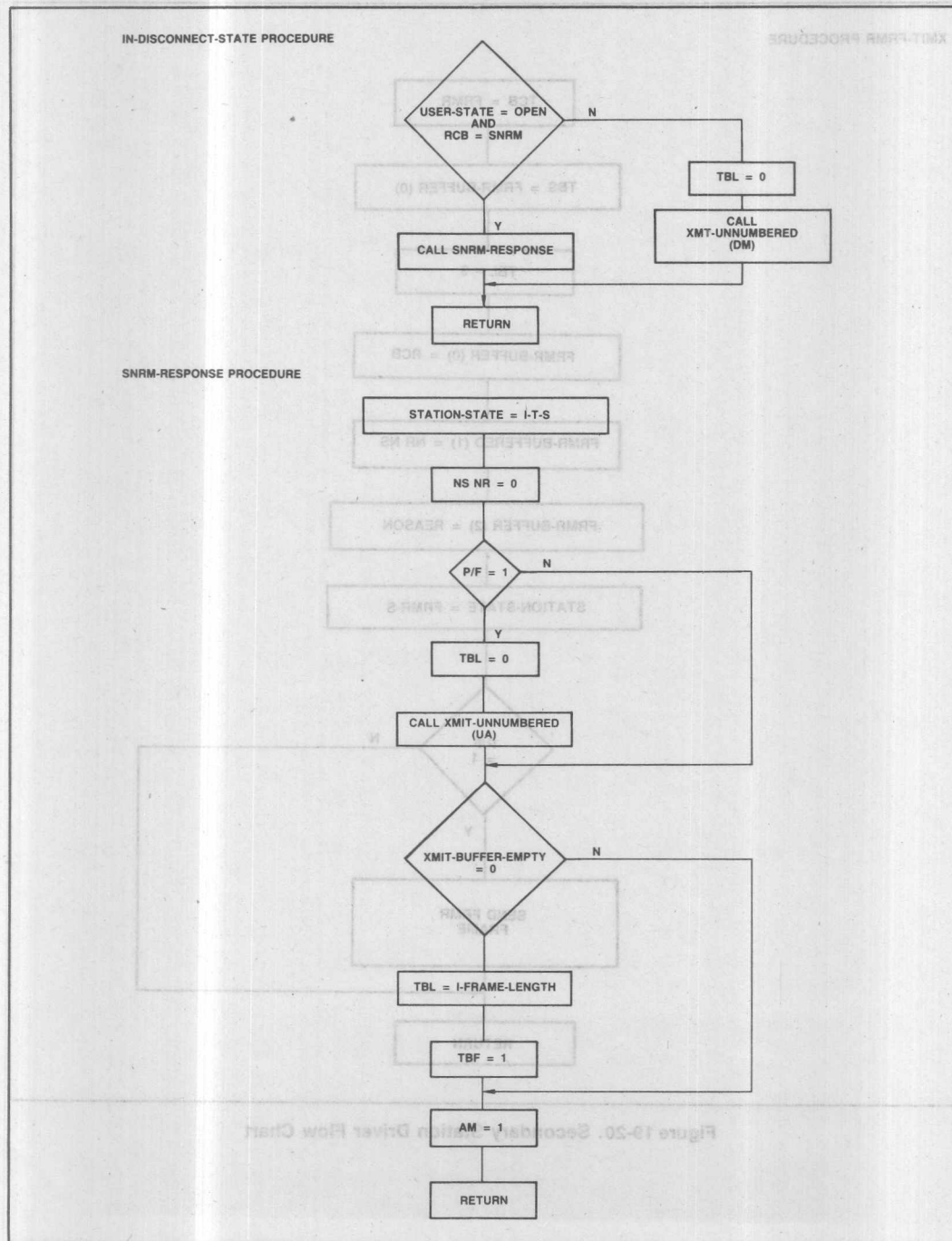


Figure 19-21. Secondary Station Driver Flow Chart

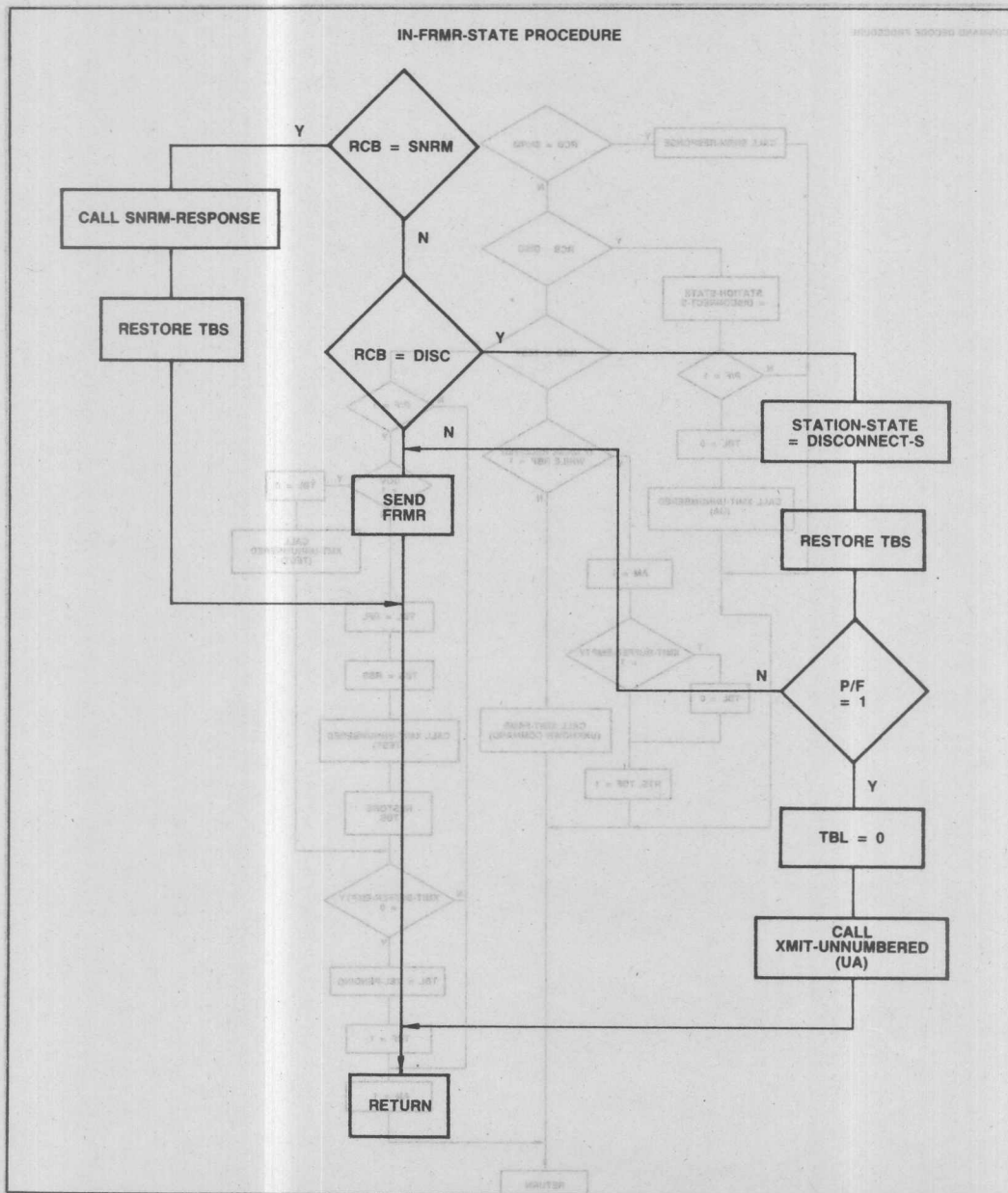


Figure 19-22 . Secondary Station Driver Flow Chart

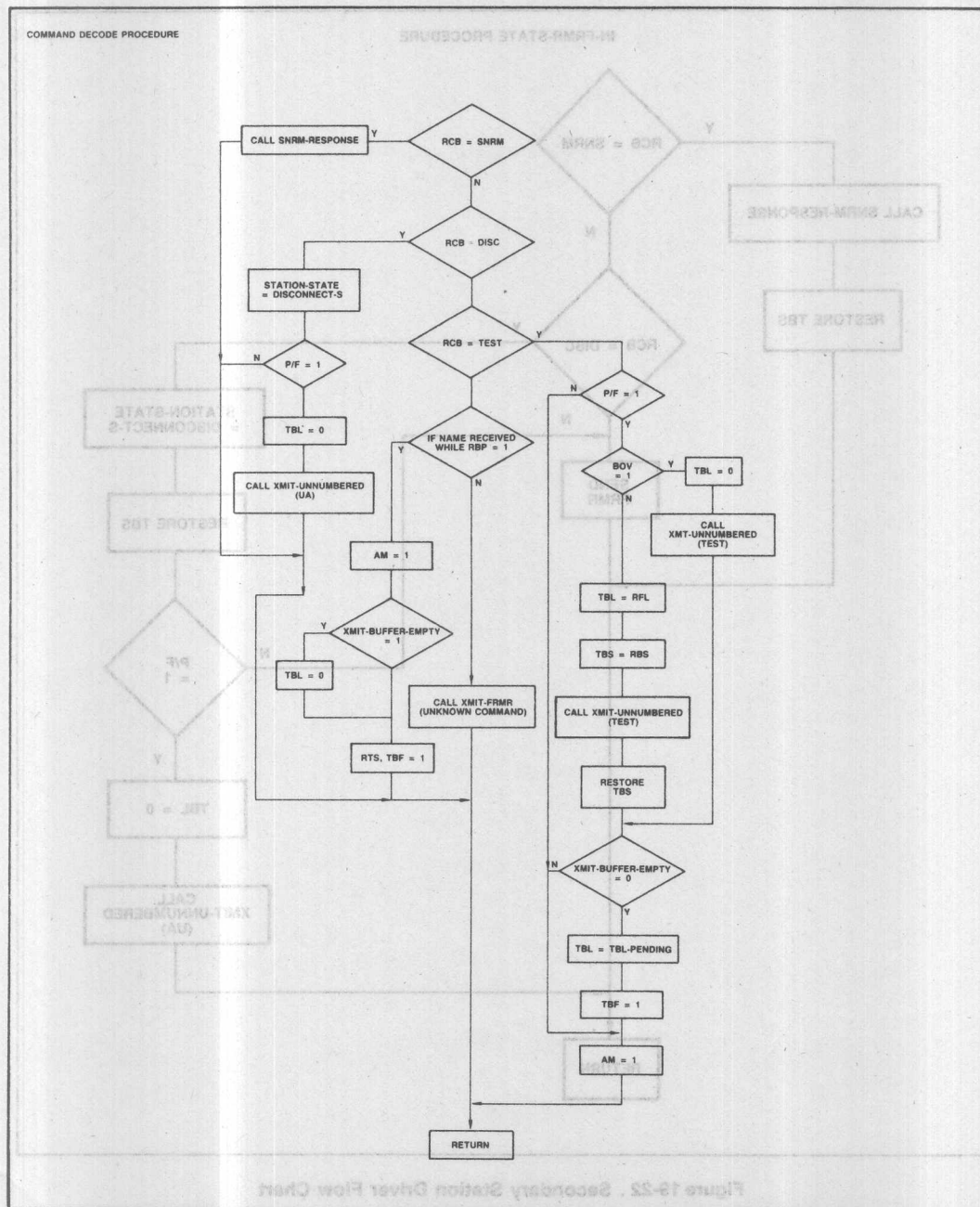


Figure 19-23. Secondary Station Driver Flow Chart

## SIU-INT PROCEDURE

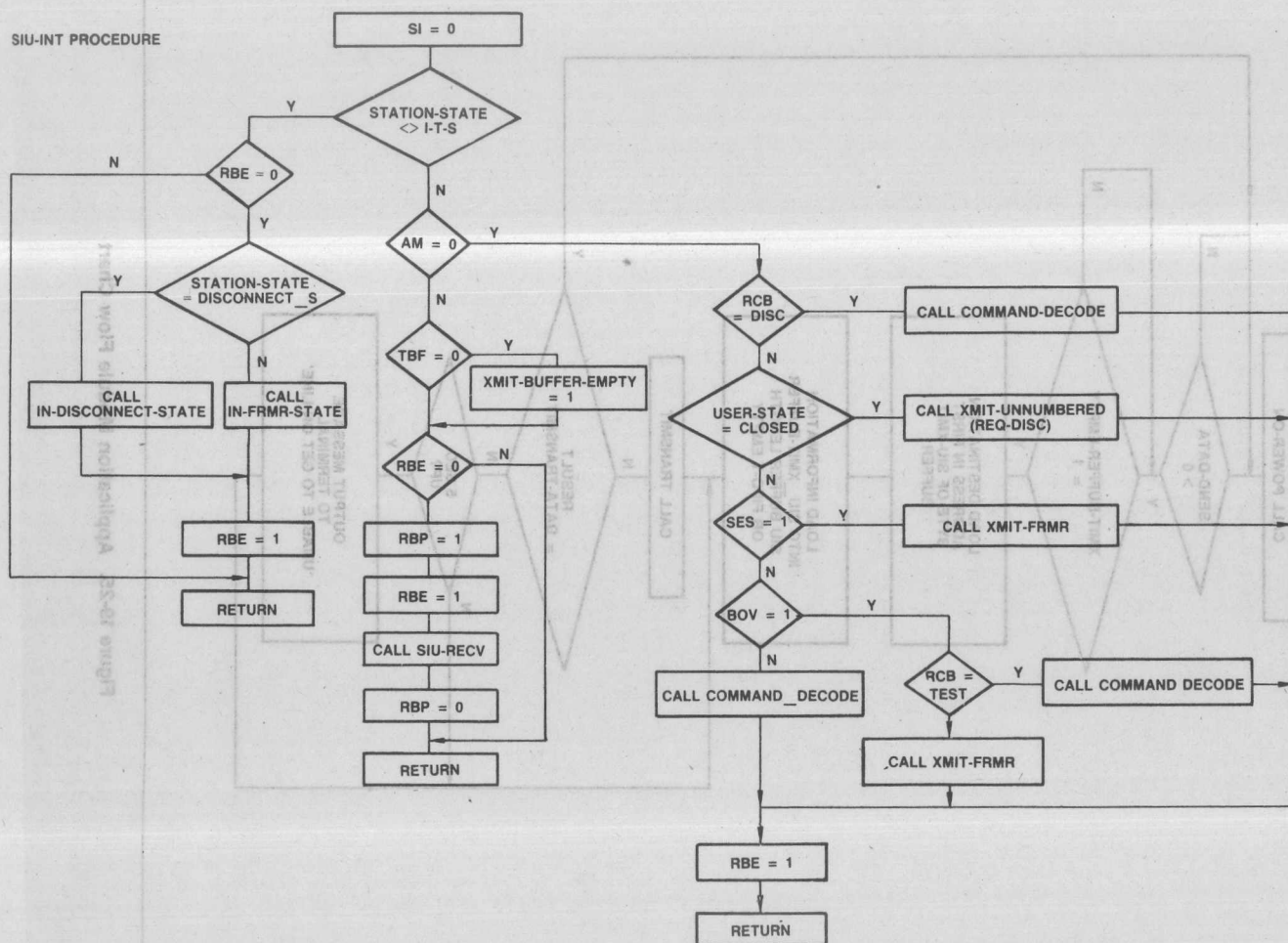


Figure 19-24. Secondary Station Driver Flow Chart



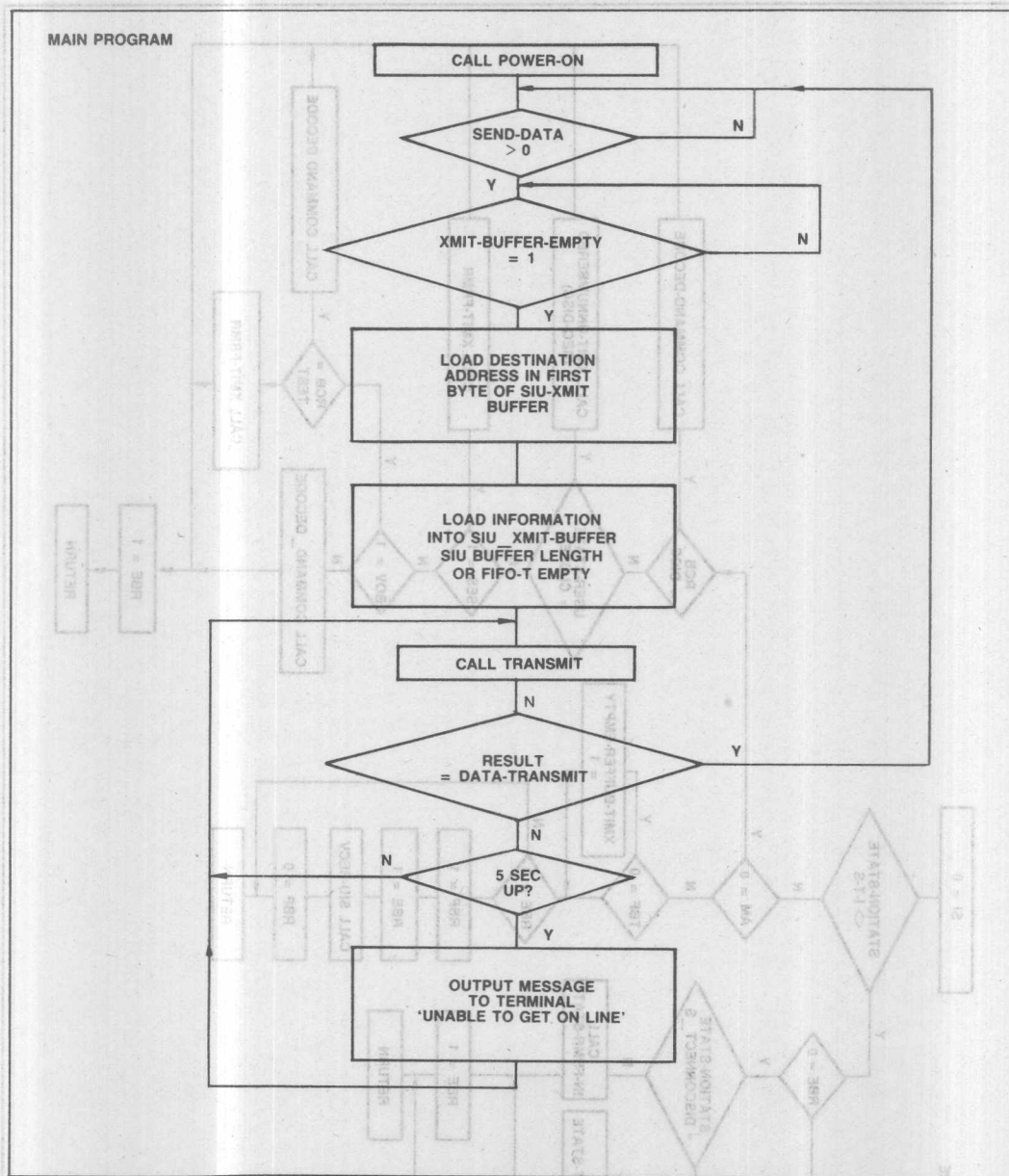
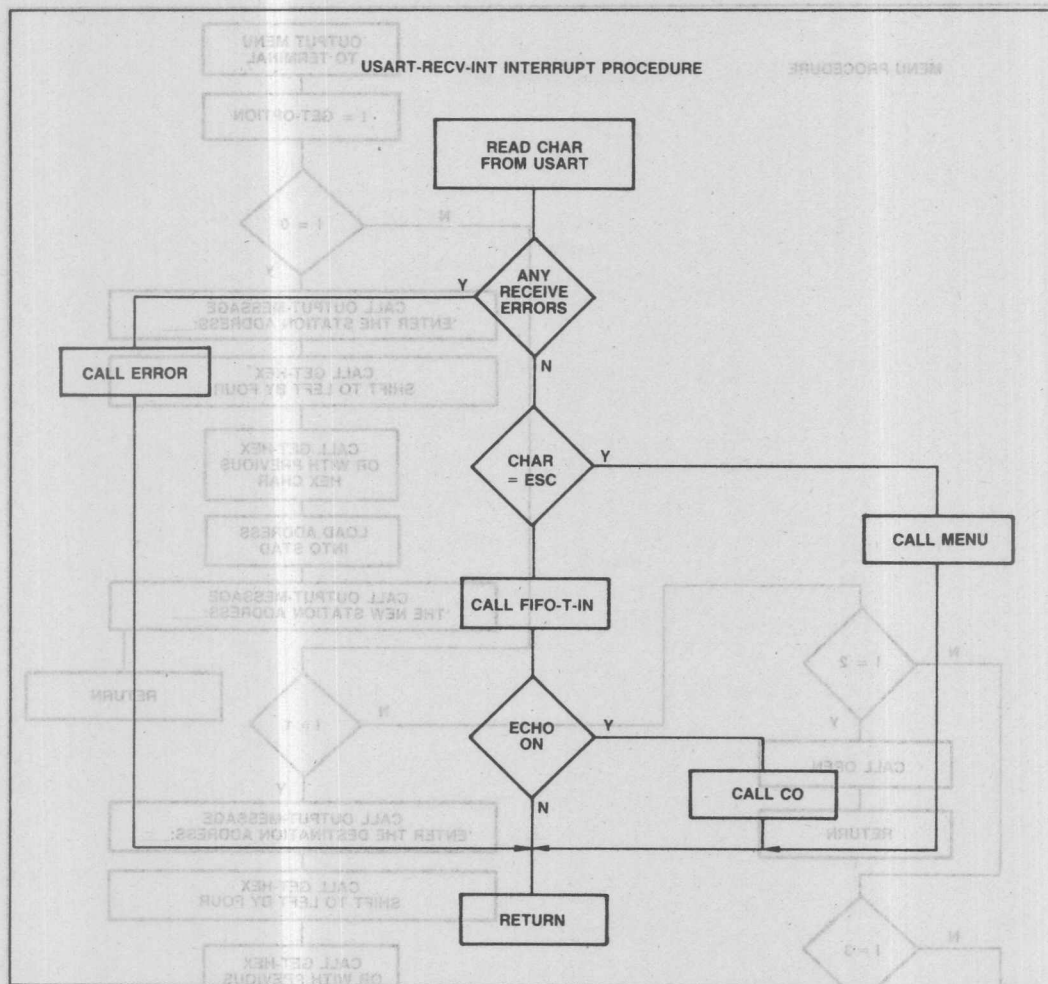


Figure 19-25. Application Module Flow Chart



**Figure 19-26. Application Module Flow Chart**

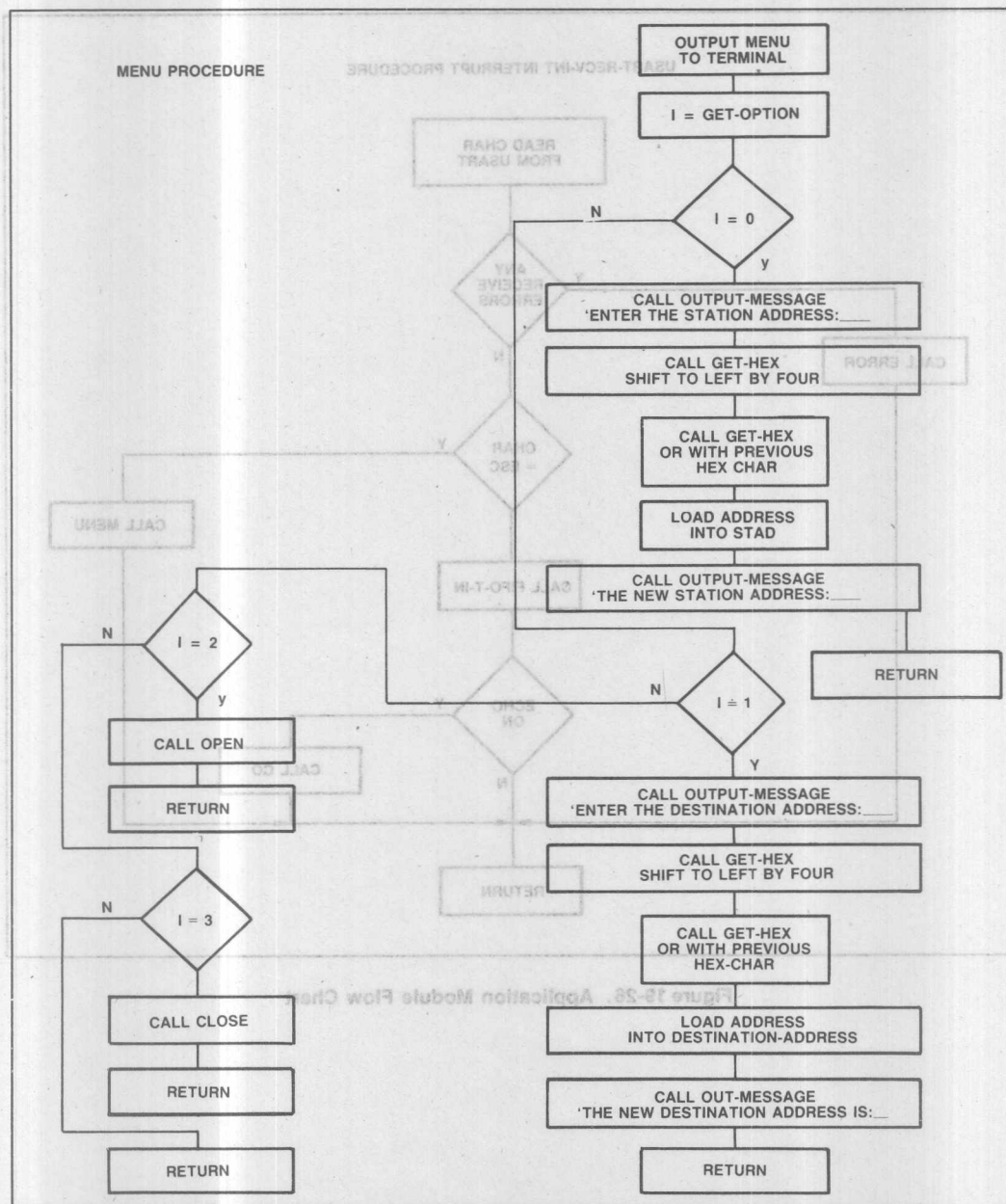


Figure 19-27. Application Module Flow Chart

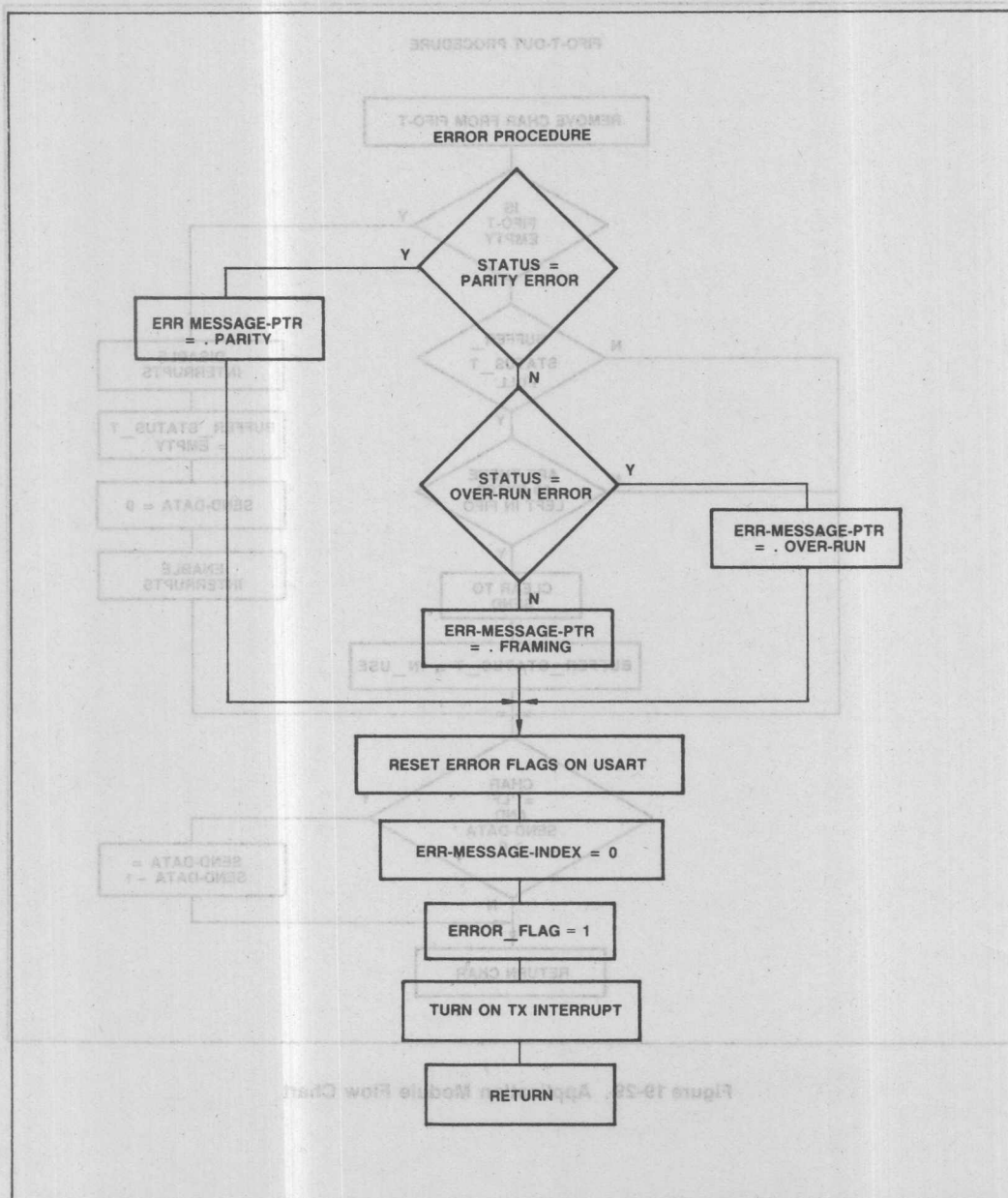


Figure 19-28 Application Module Flow Chart



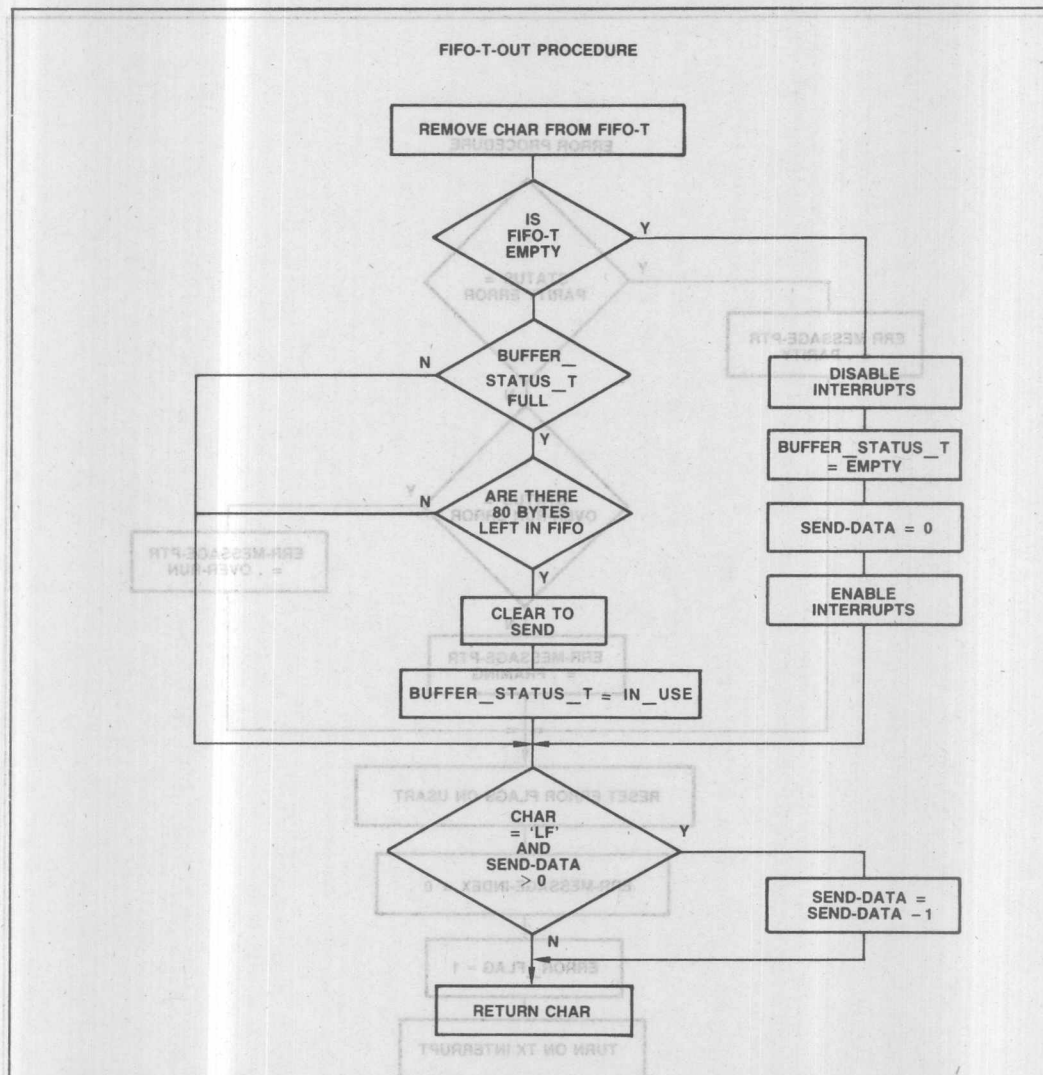


Figure 19-29. Application Module Flow Chart

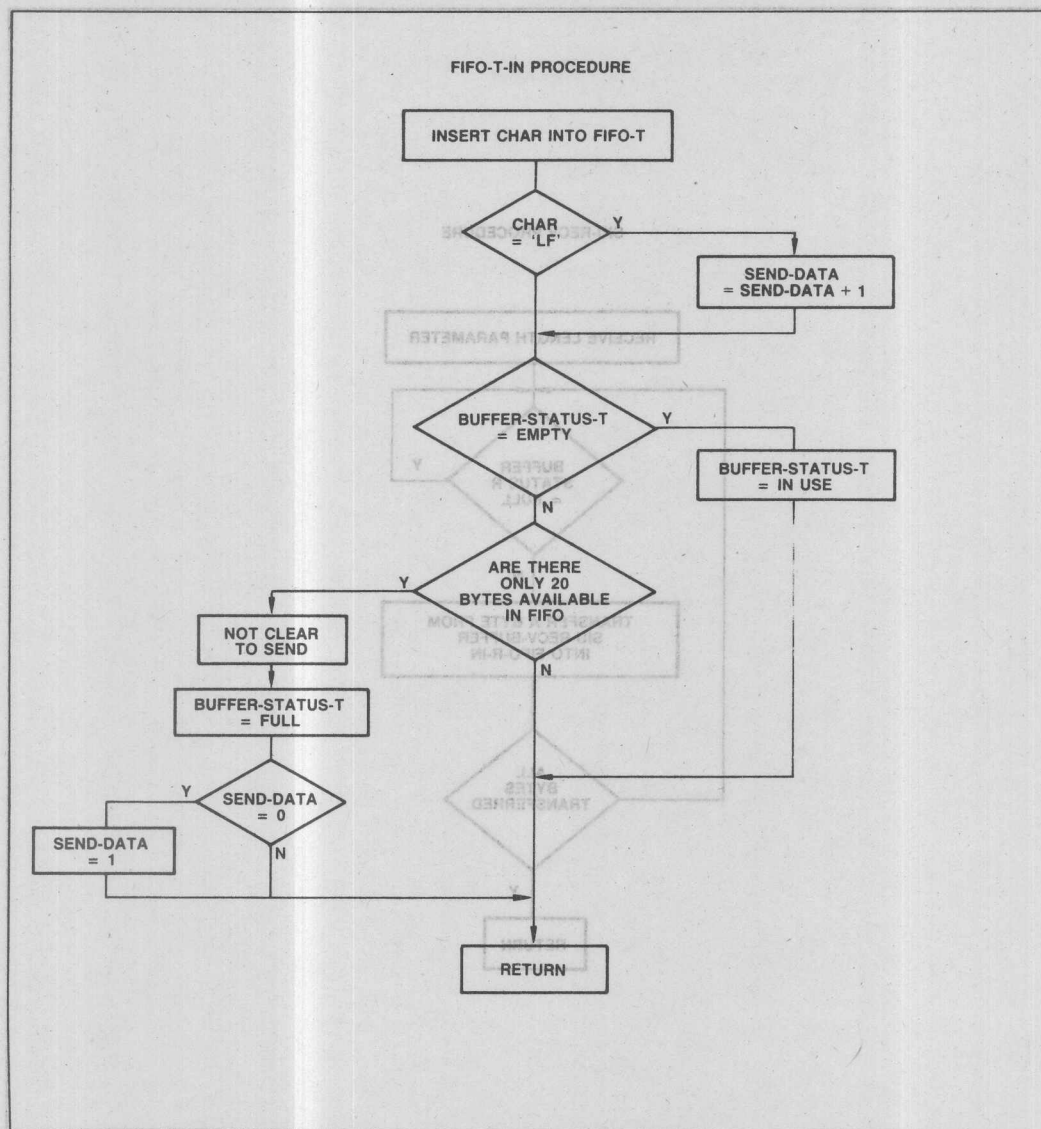


Figure 19-30. Application Module Flow Chart

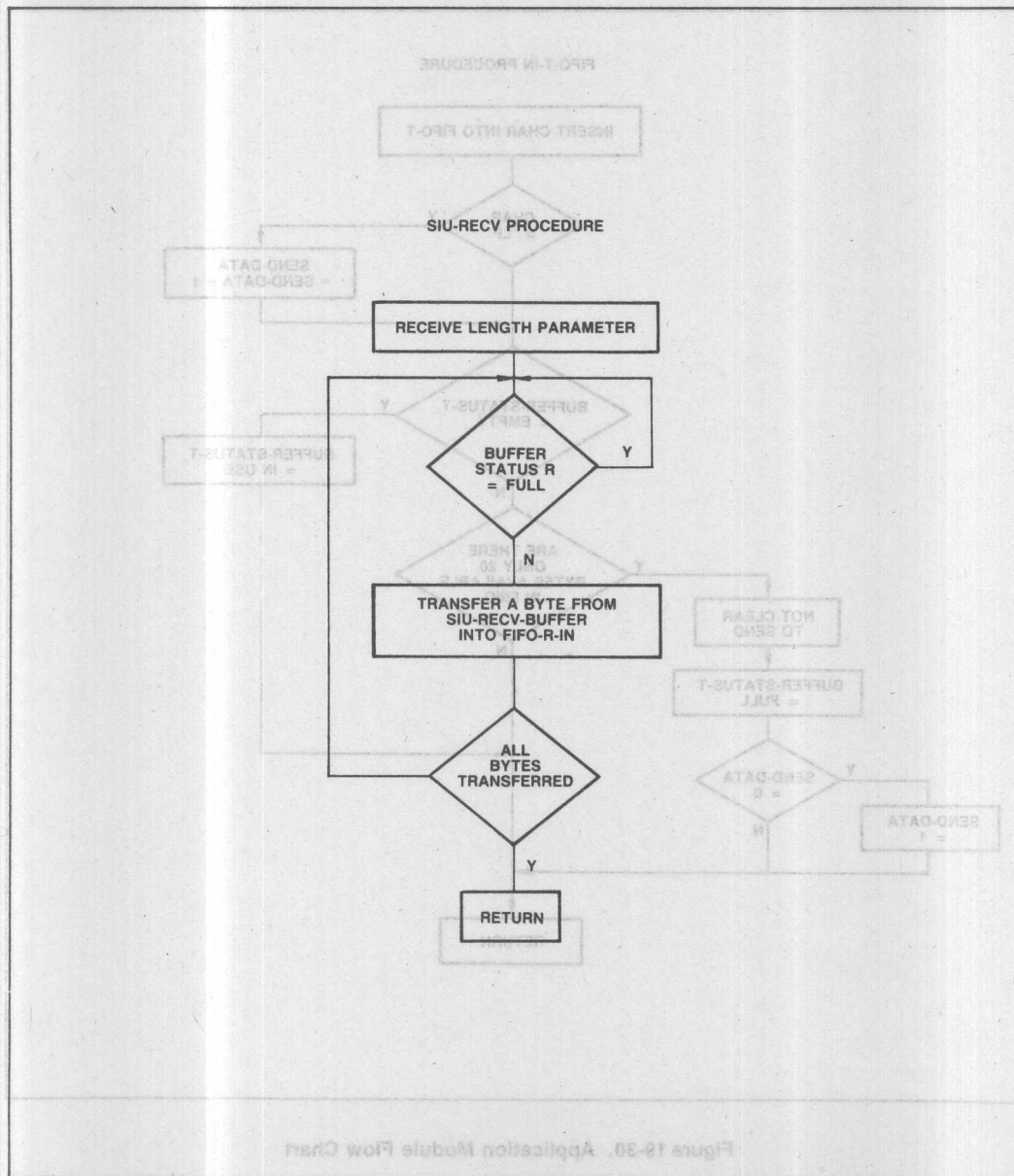


Figure 19-31. Application Module Flow Chart

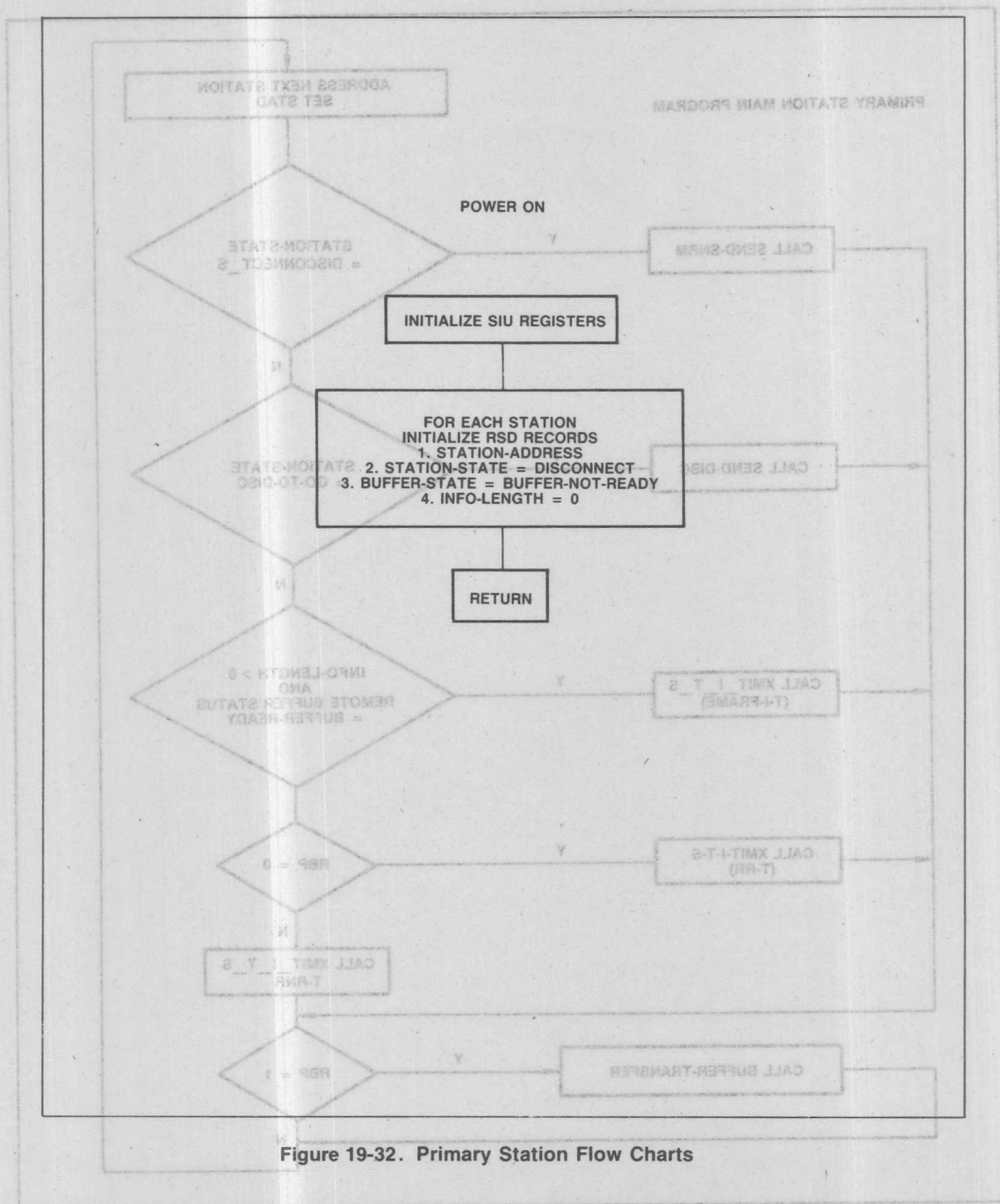


Figure 19-32. Primary Station Flow Charts

Figure 19-33. Primary Station Flow Charts



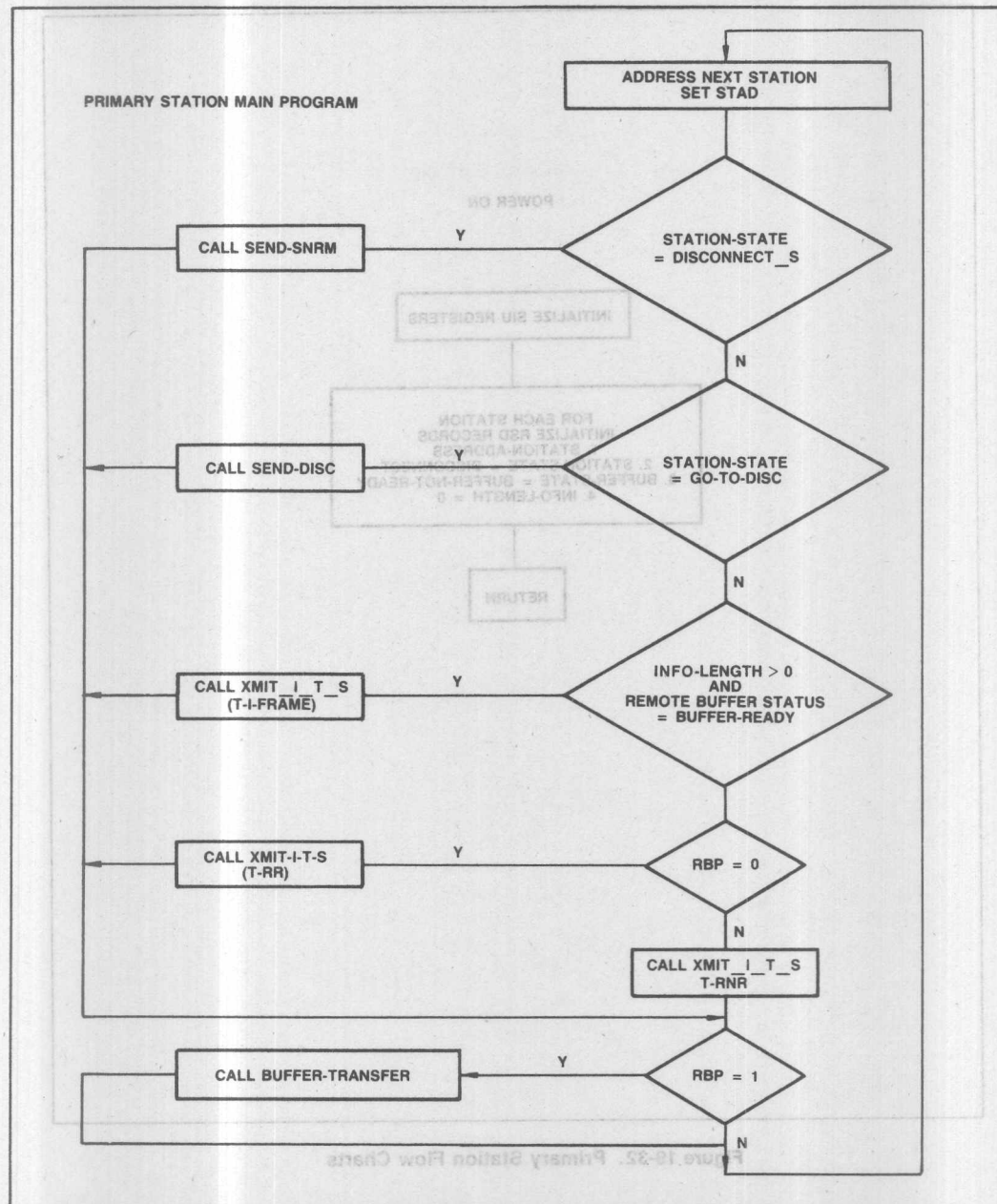


Figure 19-33. Primary Station Flow Charts

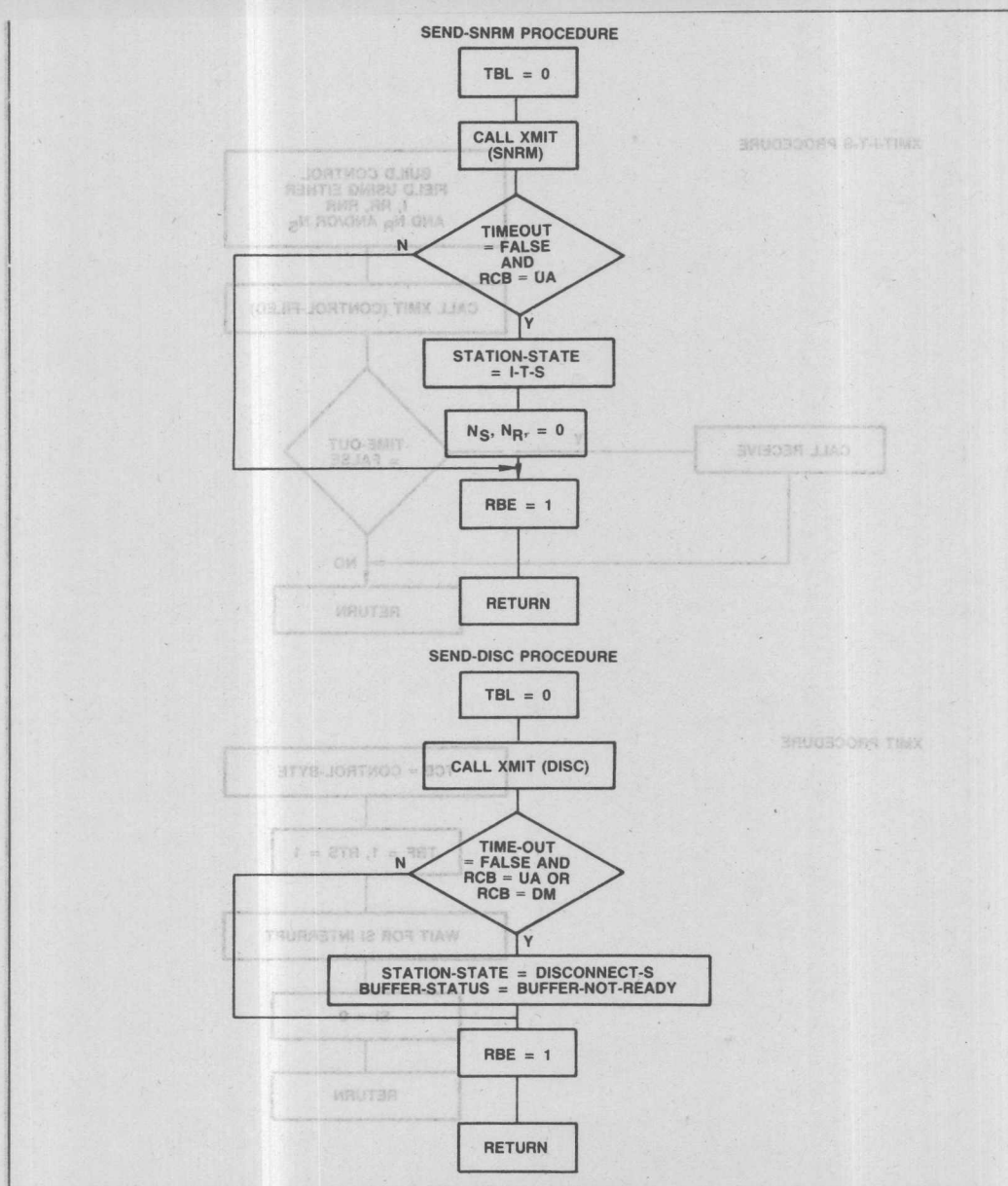


Figure 19-34. Primary Station Flow Charts

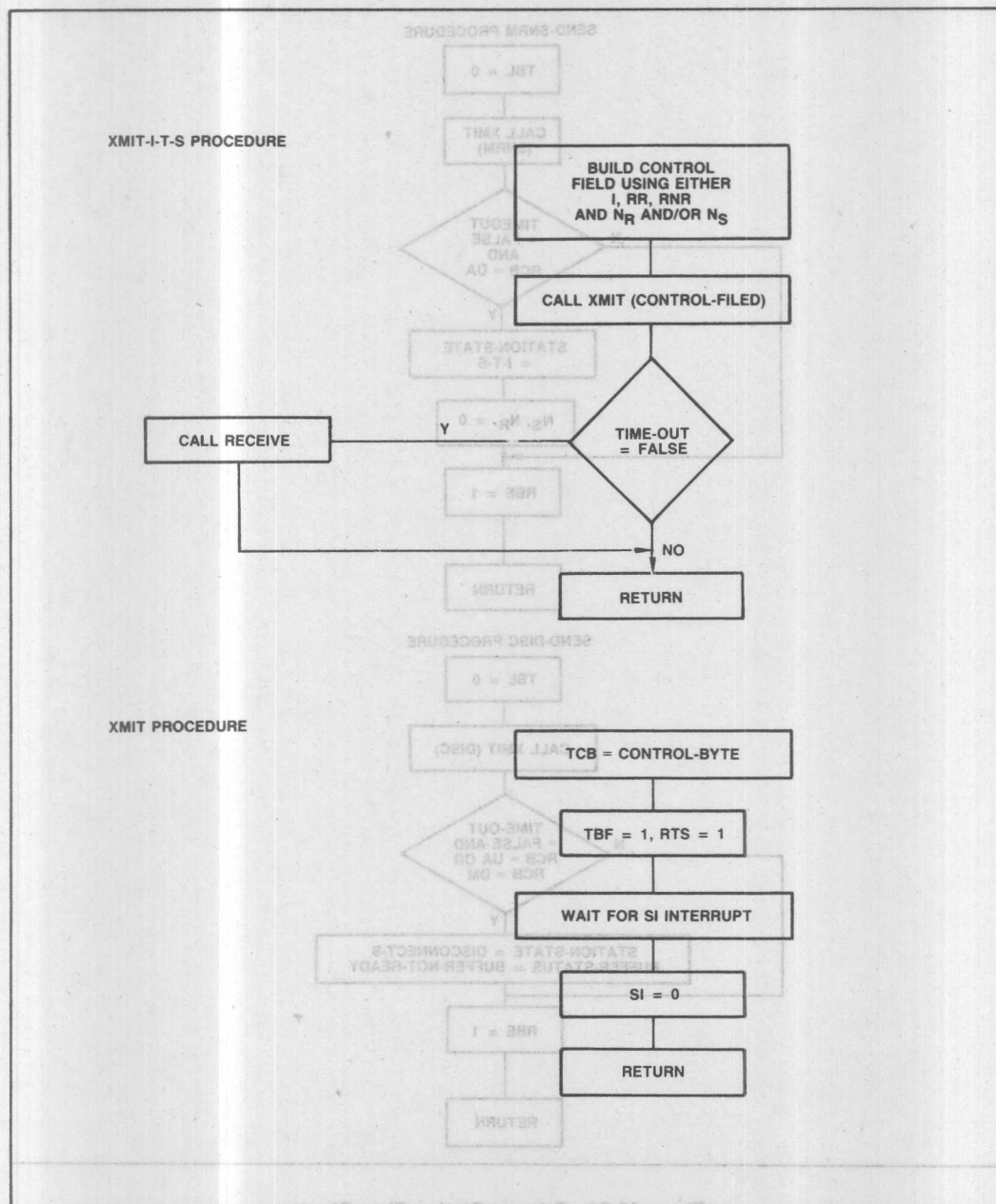


Figure 19-35. Primary Station Flow Charts

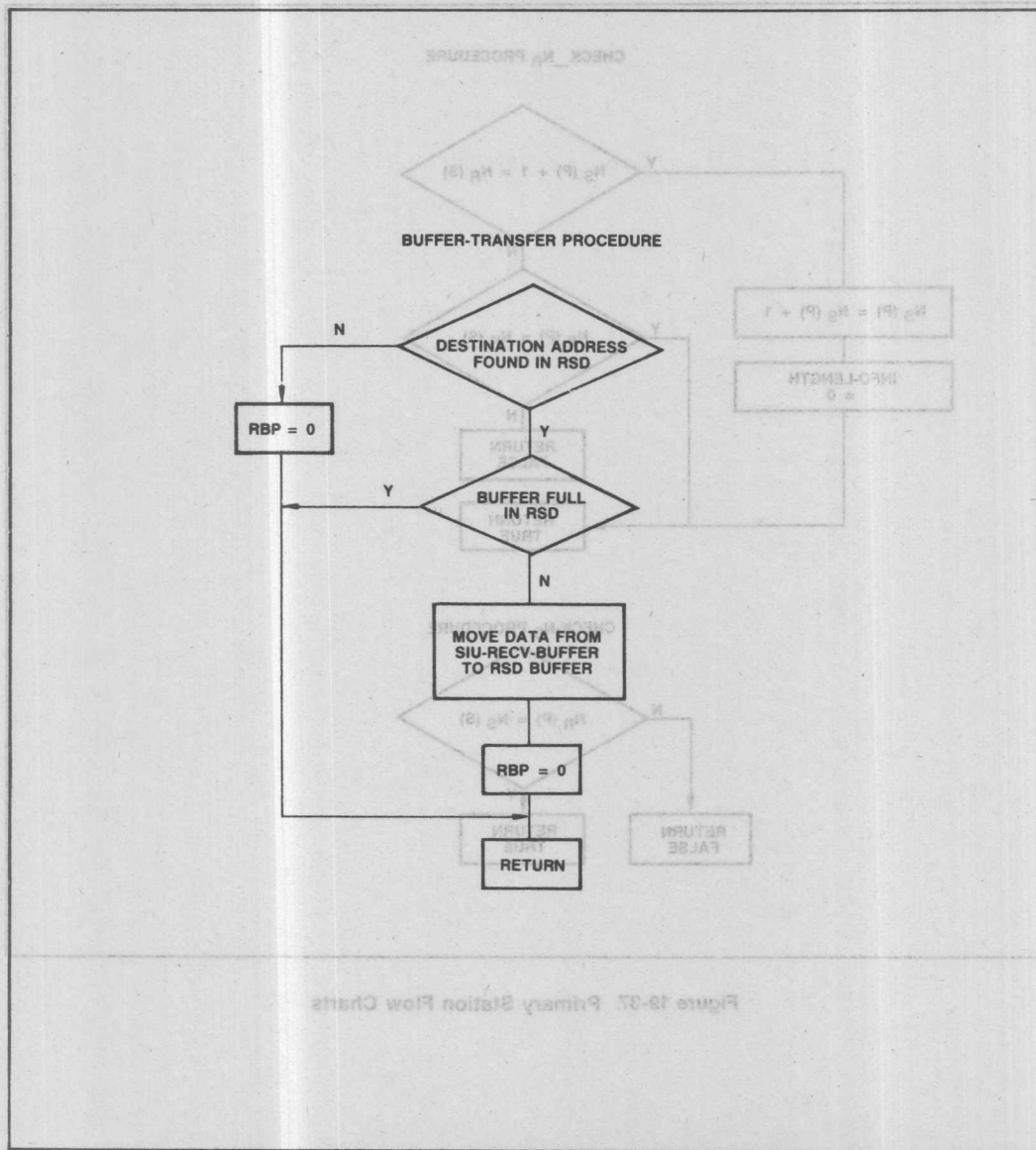


Figure 19-36. Primary Station Flow Charts



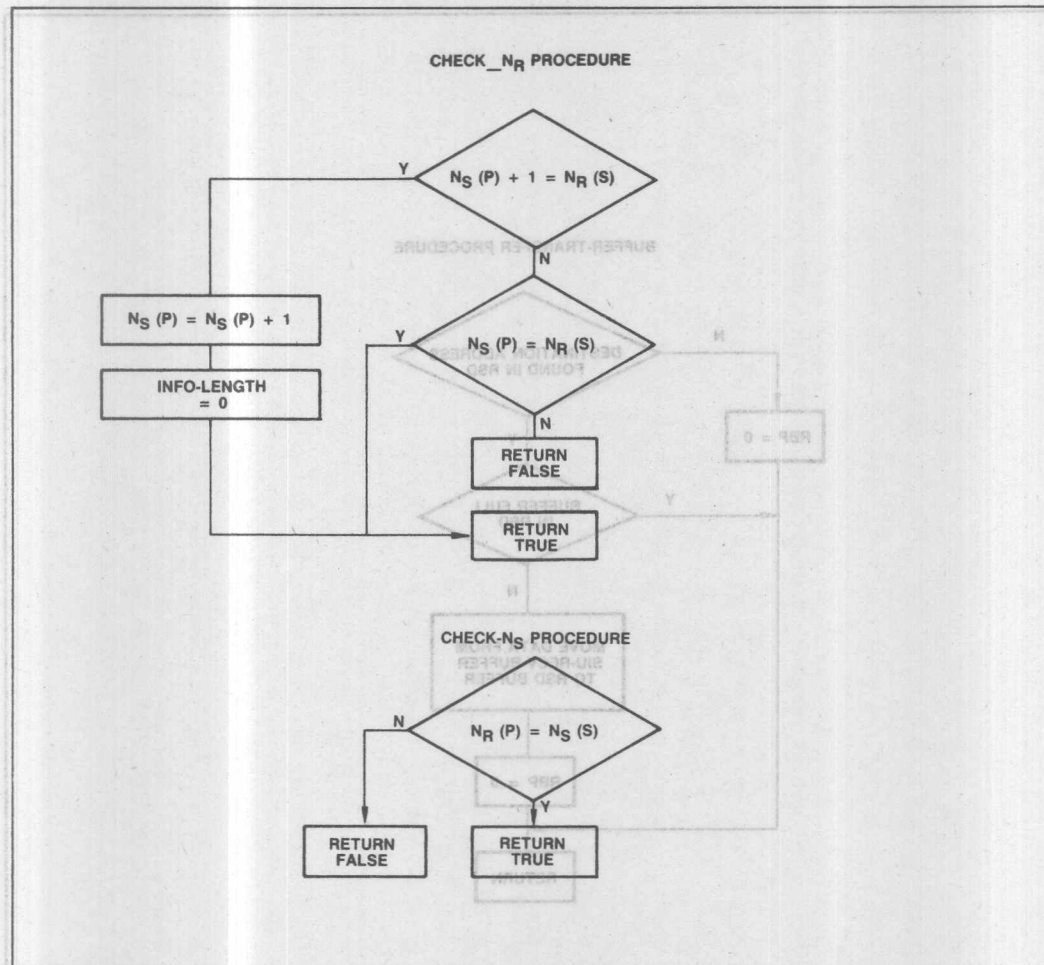
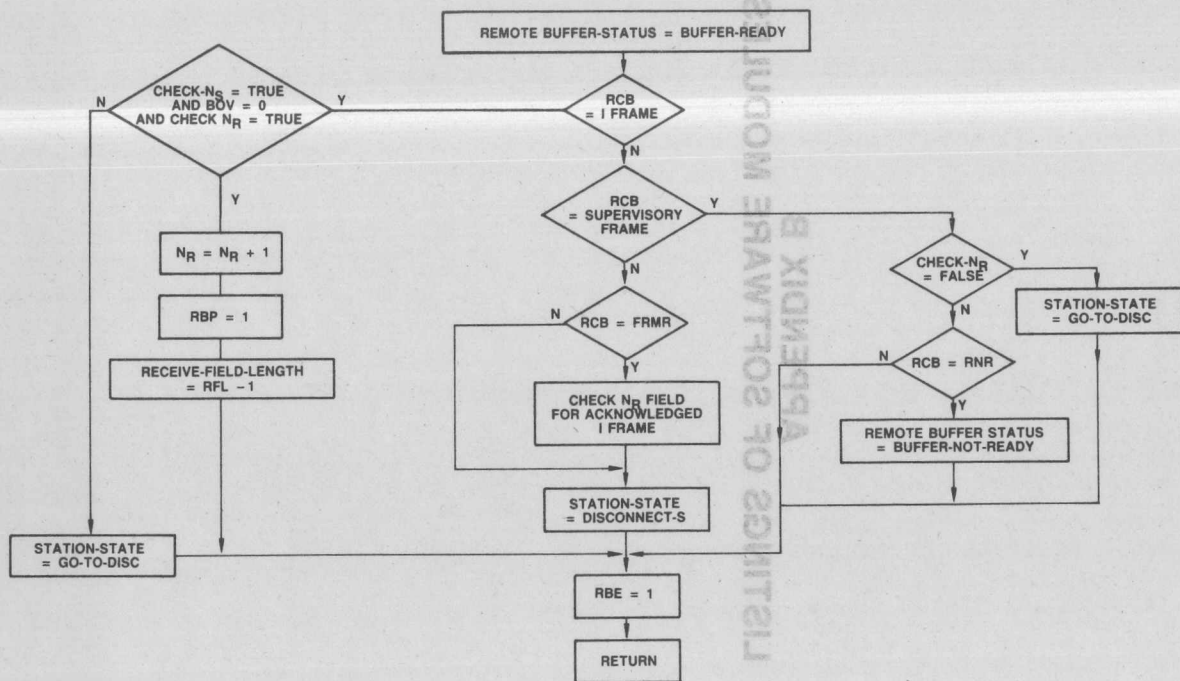


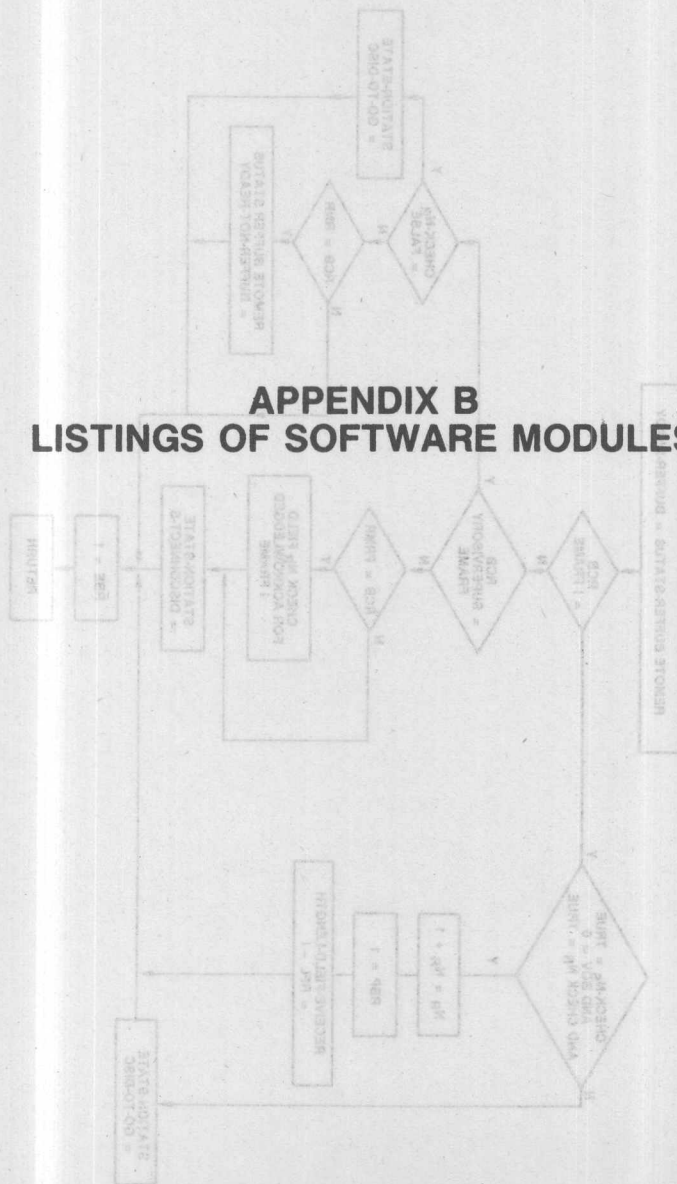
Figure 19-37. Primary Station Flow Charts

Figure 19-38. Primary Station Flow Charts



## APPENDIX B

### LISTINGS OF SOFTWARE MODULES



COMPILER INVOKED BY: : F2: PLM51 : F2: APNOTE.SRC

19-53



```

/* Variables */
USER_STATE      BYTE    AUXILIARY,
STATION_STATE   BYTE    AUXILIARY,
I_FRAME_LENGTH  BYTE    AUXILIARY,

/* Buffers */
BUFFER_LENGTH    LIT     '60',
SIU_XMIT_BUFFER(BUFFER_LENGTH)  BYTE    PUBLIC  IDATA,
SIU_RECV_BUFFER(BUFFER_LENGTH)  BYTE    PUBLIC,
FRMR_BUFFER(3)   BYTE,

/* Flags */
XMIT_BUFFER_EMPTY  BIT PUBLIC,

7 2  SIU_RECV: PROCEDURE (LENGTH) EXTERNAL;
8 2  DECLARE LENGTH BYTE;
9 1  END SIU_RECV;

10 2  OPEN: PROCEDURE PUBLIC USING 2;
11 2  USER_STATE=OPEN_S;
12 1  END OPEN;

13 2  CLOSE: PROCEDURE PUBLIC USING 2;
14 2  APM=0;
15 2  USER_STATE=CLOSED_S;
16 1  END CLOSE;

17 2  POWER_ON_D: PROCEDURE / PUBLIC USING 0;

18 2  USER_STATE=CLOSED_S;
19 2  STATION_STATE=DISCONNECT_S;
20 2  TBS= SIU_XMIT_BUFFER(0);
21 2  RBS= SIU_RECV_BUFFER(0);
22 2  RBL=BUFFER_LENGTH;
23 2  RBE=1; /* Enable the SIU's receiver */
24 2  XMIT_BUFFER_EMPTY=1;

25 1  END POWER_ON_D;

26 2  TRANSMIT: PROCEDURE (XMIT_BUFFER_LENGTH) BYTE PUBLIC USING 0;
/* User must check XMIT_BUFFER_EMPTY flag before calling this procedure */

27 2  DECLARE XMIT_BUFFER_LENGTH  BYTE,
1  STATUS  BYTE AUXILIARY,
STATUS  BYTE AUXILIARY;

28 2  IF USER_STATE=CLOSED_S
29 2  THEN STATUS=USER_STATE_CLOSED;
30 2  ELSE IF STATION_STATE=DISCONNECT_S
31 2  THEN STATUS=LINK_DISCONNECTED;
32 2  ELSE IF XMIT_BUFFER_LENGTH>BUFFER_LENGTH
33 2  THEN STATUS=OVERFLOW;
34 3  ELSE DO;

```

```

35 3      XMIT_BUFFER_EMPTY=0;
36 3      TBL=XMIT_BUFFER_LENGTH;
37 3      I_FRAME_LENGTH=XMIT_BUFFER_LENGTH; /* Store length in case station
                                           is reset by FRMR, SNRM etc. */
38 3      TBF=1;
39 3      STATUS=DATA_TRANSMITTED;
40 3      END;
41 2      RETURN STATUS;
42 1      END TRANSMIT;

43 2      XMIT_UNNUMBERED: PROCEDURE (CONTROL_BYTE);
44 2          DECLARE CONTROL_BYTE BYTE;
45 2          TCB=CONTROL_BYTE;
46 2          TBF=1;
47 2          RTS=1;
48 3          DO WHILE NOT SI;
49 3              END;
50 2          SI=0;

51 1      END XMIT_UNNUMBERED;

52 2      SNRM_RESPONSE: PROCEDURE ;
53 2          STATION_STATE=I_T_S;
54 2          NSNR=0;
55 2          IF (RCB AND 10H) <> 0 /* Respond if polled */
56 2              THEN DO;
57 3                  TBL=0;
58 3                  CALL XMIT_UNNUMBERED(UA);
59 3                  END;
60 2          IF XMIT_BUFFER_EMPTY=0 /* If an I frame was left pending transmission
                                   then restore it */
61 2              THEN DO;
62 3                  TBL=I_FRAME_LENGTH;
63 3                  TBF=1;
64 3                  END;
65 2          AM=1;

66 1      END SNRM_RESPONSE;

67 2      XMIT_FRMR: PROCEDURE (REASON);
68 2          DECLARE REASON BYTE;
69 2          TCB=FRMR;
70 2          TBS=. FRMR_BUFFER(0);
71 2          TBL=3;
72 2          FRMR_BUFFER(0)=RCB;
73 2          /* Swap nibbles in NSNR */
74 3          FRMR_BUFFER(1)=(SHL((NSNR AND 0EH),4) OR SHR((NSNR AND 0EH),4));
75 3          DO CASE REASON;
76 3              FRMR_BUFFER(2)=01H; /* UNASSIGNED_C */

```

```

76 3      FRMR_BUFFER(2)=02H; /* NO_I_FIELD_ALLOWED */
77 3      FRMR_BUFFER(2)=04H; /* BUFF_OVERRUN */
78 3      FRMR_BUFFER(2)=08H; /* SES_ERR */
79 3      END;

80 2      STATION_STATE=FRMR_S;

81 2      IF (RCB AND 10H) <> 0
      THEN DO;
83 3          TBF=1;
84 3          RTS=1;
85 4          DO WHILE NOT SI;
86 4              END;
87 3          SI=0;
88 3          END;
89 1      END XMIT_FRMR;

90 2      IN_DISCONNECT_STATE: PROCEDURE /* Called from SIU_INT procedure */
91 2          IF ((USER_STATE=OPEN_S) AND ((RCB AND 0EFH)=SNRM))
      THEN CALL SNRM_RESPONSE;
93 2          ELSE IF (RCB AND 10H) <> 0
      THEN DO;
95 3              TBL=0;
96 3              CALL XMIT_UNNUMBERED(DM);
97 3              END;
98 1      END IN_DISCONNECT_STATE;

99 2      IN_FRMR_STATE: PROCEDURE /* Called by SIU_INT when a frame has been received
      when in the FRMR state */
100 2          IF (RCB AND 0EFH)=SNRM
      THEN DO;
102 3              CALL SNRM_RESPONSE;
103 3              TBS= SIU_XMIT_BUFFER(0); /* Restore transmit buffer start address */
104 3              END;
105 2          ELSE IF (RCB AND 0EFH)=DISC
      THEN DO;
107 3              STATION_STATE=DISCONNECT_S;
108 3              TBS= SIU_XMIT_BUFFER(0); /* Restore transmit buffer start address */
109 3              IF (RCB AND 10H) <> 0
      THEN DO;
111 4                  TBL=0;
112 4                  CALL XMIT_UNNUMBERED(UA);
113 4                  END;
114 3              END;
115 3          ELSE DO; /* Receive control byte is something other than DISC or SNRM */
116 3              IF (RCB AND 10H) <> 0
      THEN DO;
118 4                  TBF=1;
119 4                  RTS=1;

```

PL/M-51 COMPILER RUPI-44 Secondary Station Driver

20:24:47 09/20/83 PAGE 5

```

120 5          DO WHILE NOT SI;
121 5          END;
122 4          END;
123 3          END;

124 1      END IN_FMRM_STATE;

125 2      COMMAND_DECODE: PROCEDURE ;

126 2          IF (RCB AND OEFH)=SNRM
              THEN CALL SNRM_RESPONSE;

128 2          ELSE IF (RCB AND OEFH)=DISC
              THEN DO;
130 3              STATION_STATE=DISCONNECT_S;
131 3              IF (RCB AND IOH)<0
                  THEN DO;
133 4                  TBL=0;
134 4                  CALL XMIT_UNNUMBERED(UA);
135 4                  END;
136 3              END;

137 2          ELSE IF (RCB AND OEFH)=TEST
              THEN DO;
139 3              IF (RCB AND IOH)>0 /* Respond if polled */
                  THEN DO; /* FOR BOV=1, SEND THE TEST RESPONSE WITHOUT AN I FIELD */
141 4                  IF (BOV=1)
                      THEN DO;
143 5                      TBL=0;
144 5                      CALL XMIT_UNNUMBERED(TEST OR IOH);
145 5                      END;
146 5                      ELSE DO; /* If no BOV, send received I field back to primary */
147 5                          TBL=RFL;
148 5                          TBS=RBS;
149 5                          CALL XMIT_UNNUMBERED(TEST OR IOH);
150 5                          TBS=SIU_XMIT_BUFFER(0); /* Restore TBS */
151 5                          END;
                      /* If an I frame was pending, set it up again */
152 4                      IF XMIT_BUFFER_EMPTY=0
                          THEN DO;
154 5                          TBL=I_FRAME_LENGTH;
155 5                          TBF=1;
156 5                          END;
157 4                      END;
158 3                      AM=1;
159 3                      END;

160 2          ELSE IF (RCB AND OIH) = 0 /* Kicked out of the AUTO mode because
              an I frame was received while RPB = 1 */
              THEN DO;
162 3              AM = 1;
163 3              IF XMIT_BUFFER_EMPTY = 1
                  THEN TBL = 0;
165 3              TBF = 1; /* Send an AUTO mode response */

```



```

166 3          RTS = 1;
167 3          END;
168 2          ELSE CALL XMIT_FRMR(UNASSIGNED_C); /* Received an undefined or not implemented command */
169 1          END COMMAND_DECODE;

170 2          SIU_INT: PROCEDURE INTERRUPT 4;
171 2          DECLARE I BYTE AUXILIARY;
172 2          SI=0;
173 2          IF STATION_STATE<> I_T_S /* Must be in NON-AUTO mode */
174 2          THEN DO;
175 3              IF RBE=0 /* Received a frame? Give response */
176 3              THEN DO;
177 5                  DO CASE STATION_STATE;
178 5                      CALL IN_DISCONNECT_STATE;
179 5                      CALL IN_FRMR_STATE;
180 5                      END;
181 4                      RBE=1;
182 4                  END;
183 3              RETURN;
184 3              END;
/* If the program reaches this point, STATION_STATE=I_T_S
which means the SIU either was, or still is in the AUTO MODE */

185 2          IF AM=0
186 2          THEN DO;
187 3              IF (RCB AND OEFH)=DISC
188 3              THEN CALL COMMAND_DECODE;
189 3              ELSE IF USER_STATE=CLOSED_S
190 3              THEN DO;
191 4                  TBL=0;
192 4                  CALL XMIT_UNNUMBERED(REG_DISC);
193 4                  END;
194 3              ELSE IF SES=1
195 3              THEN CALL XMIT_FRMR(SES_ERR);
196 3              ELSE IF BOV=1
197 3              THEN DO; /* DON'T SEND FRMR IF A TEST WAS RECEIVED */
198 4                  IF (RCB AND OEFH)=TEST
199 4                  THEN CALL COMMAND_DECODE;
200 4                  ELSE CALL XMIT_FRMR(BUFF_OVERRUN);
201 4                  END;
202 3              ELSE CALL COMMAND_DECODE;
203 3              RBE=1;
204 3              END;
205 3          ELSE DO; /* MUST STILL BE IN AUTO MODE */
206 3              IF TBF=0
207 3              THEN XMIT_BUFFER_EMPTY=1; /* TRANSMITTED A FRAME */
208 3              IF RBE=0
209 3              THEN DO;

```

```

210 4          RBP=1; /* RNR STATE */
211 4          RBE=1; /* RE-ENABLE RECEIVER */
212 4          CALL SIU_RECV(RFL);
213 4          RBP=0; /* RR STATE */
214 4          END;
215 3          END SIU_INT;
216 1          END MAIN$MOD;
217 1          END MAIN$MOD;

```

Software and application note written by Charles Yager

WARNINGS:  
4 IS THE HIGHEST USED INTERRUPT

MODULE INFORMATION:	(STATIC+OVERLAYABLE)
CODE SIZE	= 028FH 655D
CONSTANT SIZE	= 0000H 0D
DIRECT VARIABLE SIZE	= 3FH+02H 63D+ 2D
INDIRECT VARIABLE SIZE	= 3CH+00H 60D+ 0D
BIT SIZE	= 01H+00H 1D+ 0D
BIT-ADDRESSABLE SIZE	= 00H+00H 0D+ 0D
AUXILIARY VARIABLE SIZE	= 0006H 6D
MAXIMUM STACK SIZE	= 0017H 23D
REGISTER-BANK(S) USED:	0 1 2
460 LINES READ	
0 PROGRAM ERROR(S)	
END OF PL/M-51 COMPILATION	

ISIS-11 PL/M-51 V1.0

COMPILER INVOKED BY: :#2:plm51 :#2:unote:src

```

$TITLE      ('Application Module: Async/SDLC Protocol converter')
$debug
$registerbank(0)
user$mod:do:
1 1 $NOLIST
5 1 DECLARE
    LIT      LITERALLY      'LITERALLY',
    TRUE     LIT             'OFFH',
    FALSE    LIT             'OOH',
    FOREVER  LIT             'WHILE 1',
    ESC      LIT             '1BH',
    LF       LIT             '0AH',
    CR       LIT             '0DH',
    BS       LIT             '0BH',
    BEL      LIT             '07H',
    EMPTY    LIT             '00H',
    INUSE     LIT             '01H',
    FULL      LIT             '02H',
    USER_STATE_CLOSED LIT    '00H',
    LINK_DISCONNECTED LIT    '01H',
    OVERFLOW  LIT             '02H',
    DATA_TRANSMITTED LIT     '03H',

    /* BUFFERS */
    BUFFER_LENGTH LIT        '60',
    SIU_XMIT_BUFFER(BUFFER_LENGTH) BYTE EXTERNAL IDATA,
    SIU_RECV_BUFFER(BUFFER_LENGTH) BYTE EXTERNAL,
    FIFO_T(256) BYTE AUXILIARY,
    IN_PTR_T BYTE AUXILIARY,
    OUT_PTR_T BYTE AUXILIARY,
    BUFFER_STATUS_T BYTE AUXILIARY,
    FIFO_R(256) BYTE AUXILIARY,
    IN_PTR_R BYTE AUXILIARY,
    OUT_PTR_R BYTE AUXILIARY,
    BUFFER_STATUS_R BYTE AUXILIARY,

    /* Variables and Parameters */
    LENGTH BYTE AUXILIARY,
    CHAR BYTE AUXILIARY,
    I BYTE AUXILIARY,
    USART_CMD BYTE AUXILIARY,
    DESTINATION_ADDRESS BYTE AUXILIARY,
    SEND_DATA BYTE AUXILIARY,
    RESULT BYTE AUXILIARY,
    ERR_MESSAGE_INDEX BYTE AUXILIARY,
    ERR_MESSAGE_PTR WORD AUXILIARY,

    /* Messages Sent to the Terminal */
    PARITY(*) BYTE CONSTANT(LF,CR,'Parity Error Detected',LF,CR,OOH),
    FRAME(*) BYTE CONSTANT(LF,CR,'Framing Error Detected',LF,CR,OOH),

```

```

OVER_RUN(*) BYTE CONSTANT(LF,CR,'Overrun Error Detected',LF,CR,0),
LINK(*) BYTE CONSTANT(LF,CR,'Unable to Get Online',LF,CR,00H),
DEST_ADDR(*) BYTE CONSTANT(CR,LF,LF,
    'Enter the destination address: ',BS,BS,0),
D_ADDR_ACK(*) BYTE CONSTANT(CR,LF,LF,
    'The new destination address is ',0),
STAT_ADDR(*) BYTE CONSTANT(CR,LF,LF,
    'Enter the station address: ',BS,BS,0),
S_ADDR_ACK(*) BYTE CONSTANT(CR,LF,LF,
    'The new station address is ',0),
ADDR_ACK_FIN(*) BYTE CONSTANT('H',CR,LF,LF,0),

SIGN_ON(*) BYTE CONSTANT(CR,LF,LF,
    '(\') RUPI-44 Secondary Station',CR,LF,
    '\',CR,LF,LF,
    '1 - Set the Station Address',LF,CR,
    '2 - Set the Destination Address',CR,LF,
    '3 - Go Online',CR,LF,
    '4 - Go Offline',CR,LF,
    '5 - Return to terminal mode',CR,LF,LF,
    'Enter option: ',BS,0),
FIN(*) BYTE CONSTANT(CR,LF,LF,0),

/* Characters Received From the Terminal */
HEX_TABLE(17) BYTE CONSTANT('0123456789ABCDEF',BEL),
MENU_CHAR(6) BYTE CONSTANT('12345',BEL),

/* Flags and Bits */
XMIT_BUFFER_EMPTY    BIT    EXTERNAL, /* Semaphore for RUPI SIOU Transmit Buffer */
STOP_BIT             BIT    AT(147) REG, /* Terminal parameters */
ECHO                 BIT    AT(0B4H) REG,
WAIT                 BIT, /* Timeout flag */
ERROR_FLAG           BIT, /* Error message Flag */

/* Peripheral Addresses */
USART_STATUS          BYTE    AT(0B01H) AUXILIARY,
USART_DATA            BYTE    AT(0B00H) AUXILIARY,
TIMER_CONTROL         BYTE    AT(1003H) AUXILIARY,
TIMER_0               BYTE    AT(1000H) AUXILIARY,
TIMER_1               BYTE    AT(1001H) AUXILIARY,
TIMER_2               BYTE    AT(1002H) AUXILIARY,

/* External Procedures */

```

```

6 2 POWER_ON_D: PROCEDURE EXTERNAL;
7 1 END POWER_ON_D;

8 2 CLOSE: PROCEDURE EXTERNAL USING 2;
9 1 END CLOSE;

10 2 OPEN: PROCEDURE EXTERNAL USING 2;
11 1 END OPEN;

12 2 TRANSMIT: PROCEDURE (XMIT_BUFFER_LENGTH) BYTE EXTERNAL;
13 2 DECLARE XMIT_BUFFER_LENGTH BYTE;
14 1 END TRANSMIT;

/* Local Procedures */

15 2 TIMER_O_INT: PROCEDURE INTERRUPT 1 USING 1;
16 2 WAIT=0;
17 1 END TIMER_O_INT;

18 2 POWER_ON: PROCEDURE USING 0;

19 2 DECLARE TEMP BYTE AUXILIARY;

20 2 SMD=54H; /* Using DPLL, NRZI, PFS, TIMER 1, @ 62.5 Kbps */
21 2 TMD=21H; /* Timer 0 16 bit, Timer 1 auto reload */
22 2 TH1=0FFH;
23 2 TCON=40H;

24 2 TIMER_CONTROL=37H; /* Initialize USART's system clock; 8254 */
25 2 TIMER_0=04H;
26 2 TIMER_0=00H;
27 2 TIMER_CONTROL=77H; /* Initialize Tx, Rx */

/* Definition for dip switch tied to P1.0 to P1.6
Bit Rate 3 2 1
300 on on on
1200 on on off
2400 on off on
4800 on off off
9600 off on on
19200 off on off

Stop bit 4
1 on
2 off

Parity 6 5
off on on
odd on off
off off on
even off off

```



```

Echo      7
on         on
off        off      */

28 2      TEMP=P1 AND 07H; /* Read the dip switch to determine the bit rate */
29 2      IF TEMP>5
31 3      DO CASE TEMP;
32 4      /* 300 */ DO;
33 4      TIMER_1=83H;
34 4      TIMER_1=20H;
35 4      END;
36 4      /* 1200 */ DO;
37 4      TIMER_1=20H;
38 4      TIMER_1=05H;
39 4      END;
40 4      /* 2400 */ DO;
41 4      TIMER_1=60H;
42 4      TIMER_1=02H;
43 4      END;
44 4      /* 4800 */ DO;
45 4      TIMER_1=30H;
46 4      TIMER_1=01H;
47 4      END;
48 4      /* 9600 */ DO;
49 4      TIMER_1=65H;
50 4      TIMER_1=0;
51 4      END;
52 4      /* 19200 */ DO;
53 4      TIMER_1=33H;
54 4      TIMER_1=0;
55 4      END;
56 3      END;
57 2      USART_STATUS=0; /* Software power-on reset for 8251A */
58 2      USART_STATUS=0;
59 2      USART_STATUS=0;
60 2      USART_STATUS=40H;
61 2      TEMP=0AH; /* Determine the parity and # of stop bits */
62 2      TEMP=TEMP OR (P1 AND 30H);
63 2      IF STOP_BIT=1
65 2      THEN TEMP=TEMP OR 0C0H;
66 2      ELSE TEMP=TEMP OR 40H;
67 2      USART_STATUS=TEMP; /* USART Mode Word */
68 2      USART_STATUS, USART_CMD=27H; /*USART Command Word RTS, RxE, DTR, TxEN=1*/
69 2      STAD=OFFH;

```

```

69 2      SEND_DATA=0; /* Initialize Flags */
70 2      IN_PTR_T, OUT_PTR_T, IN_PTR_R, OUT_PTR_R = 0; /* Initialize FIFO PTRs */
71 2      BUFFER_STATUS_T, BUFFER_STATUS_R= EMPTY;
72 2      CALL POWER_ON_D;
73 2      IP=01H; /* USART's RxRdy is the highest priority */
74 2      IE=93H; /* Both external interrupts are level triggered */
75 2      ERROR_FLAG=0;
76 1      END POWER_ON;
77 2      FIFO_R_IN: PROCEDURE (CHAR) USING 1;
78 2      DECLARE CHAR BYTE;
79 2      FIFO_R(IN_PTR_R)=CHAR;
80 2      IN_PTR_R=IN_PTR_R+1;
81 2      IF BUFFER_STATUS_R=EMPTY
82 3      THEN DO;
83 3          EA=0;
84 3          BUFFER_STATUS_R=INUSE;
85 3          EX1=1; /* Enable USART's TxD interrupt */
86 3          EA=1;
87 3      END;
88 2      ELSE IF ((BUFFER_STATUS_R=INUSE) AND (IN_PTR_R=OUT_PTR_R))
89 3      THEN BUFFER_STATUS_R=FULL;
90 1      END FIFO_R_IN;
91 2      FIFO_R_OUT: PROCEDURE BYTE USING 1;
92 2      DECLARE CHAR BYTE AUXILIARY;
93 2      CHAR=FIFO_R(OUT_PTR_R);
94 2      OUT_PTR_R=OUT_PTR_R+1;
95 2      IF OUT_PTR_R=IN_PTR_R
96 3      THEN DO;
97 3          EX1=0; /* Shut off TxD interrupt */
98 3          BUFFER_STATUS_R=EMPTY;
99 3      END;
100 2      ELSE IF ((BUFFER_STATUS_R=FULL) AND (OUT_PTR_R-20=IN_PTR_R))
101 3      THEN BUFFER_STATUS_R=INUSE;
102 2      RETURN CHAR;
103 1      END FIFO_R_OUT;
104 2      USART_XMIT_INT: PROCEDURE INTERRUPT 2 USING 1;

```

```

105 2      DECLARE
          MESSAGE BASED ERR_MESSAGE_PTR(1)  BYTE  CONSTANT;
106 2      IF ERROR_FLAG
          THEN DO;
108 3          IF MESSAGE(ERR_MESSAGE_INDEX) <> 0 /* Then continue to send the message */
          THEN DO;
110 4              USART_DATA = MESSAGE(ERR_MESSAGE_INDEX);
111 4              ERR_MESSAGE_INDEX=ERR_MESSAGE_INDEX+1;
112 4              END;
113 4          ELSE DO; /* If message is done reset ERROR_FLAG and shut off interrupt if FIFO is empty */
114 4              ERROR_FLAG=0;
115 4              IF BUFFER_STATUS_R = EMPTY
                  THEN EX1=0;
117 4              END;
118 3      END;
119 2      ELSE USART_DATA=FIFO_R_OUT;
120 1      END USART_XMIT_INT;

121 2      SIU_RECV: PROCEDURE (LENGTH) PUBLIC USING 1;
122 2          DECLARE LENGTH  BYTE,
                      I      BYTE  AUXILIARY;
123 3          DO I=0 TO LENGTH-1;
124 4              DO WHILE BUFFER_STATUS_R=FULL; /* Check to see if fifo is full */
125 4              END;
126 3              CALL FIFO_R_IN(SIU_RECV_BUFFER(I));
127 3          END;
128 1      END SIU_RECV;

129 2      FIFO_T_IN: PROCEDURE (CHAR) USING 2;
130 2          DECLARE CHAR  BYTE;
131 2          FIFO_T(IN_PTR_T)=CHAR;
132 2          IN_PTR_T=IN_PTR_T+1;
133 2          IF CHAR=LF
              THEN SEND_DATA=SEND_DATA+1;
135 2          IF BUFFER_STATUS_T=EMPTY
              THEN BUFFER_STATUS_T=INUSE;
137 2          ELSE IF ((BUFFER_STATUS_T=INUSE) AND (IN_PTR_T+20=OUT_PTR_T))
              THEN DO; /* Stop reception using CTS */
139 3              USART_STATUS, USART_CMD=USART_CMD AND NOT(20H);
140 3              BUFFER_STATUS_T=FULL;
141 3              IF SEND_DATA=0
                  THEN SEND_DATA=1; /* If the buffer is full and no LF
                                     has been received then send data */
143 3          END;
144 1      END FIFO_T_IN;

```

```

145 2   FIFO_T_OUT: PROCEDURE BYTE ;
146 2       DECLARE CHAR   BYTE   AUXILIARY;
147 2       CHAR=FIFO_T(OUT_PTR_T);
148 2       OUT_PTR_T=OUT_PTR_T+1;
149 2       IF OUT_PTR_T=IN_PTR_T /* Then FIFO_T is empty */
150 2           THEN DO;
151 3           EA=0;
152 3           BUFFER_STATUS_T=EMPTY;
153 3           SEND_DATA=0;
154 3           EA=1;
155 3       END;
156 2       ELSE IF ((BUFFER_STATUS_T=FULL) AND (OUT_PTR_T=IN_PTR_T))
157 2           THEN DO;
158 3           USART_STATUS, USART_CMD=USART_CMD OR 20H;
159 3           BUFFER_STATUS_T=INUSE;
160 3       END;
161 2       IF (CHAR=LF AND SEND_DATA>0) THEN SEND_DATA=SEND_DATA-1;
162 2       RETURN CHAR;
163 2   END FIFO_T_OUT;
164 1
165 2   ERROR: PROCEDURE (STATUS) USING 2;
166 2       DECLARE STATUS   BYTE;
167 2       IF (STATUS AND 08H)<0
168 2           THEN ERR_MESSAGE_PTR=. PARITY;
169 2       ELSE IF (STATUS AND 10H)<0
170 2           THEN ERR_MESSAGE_PTR=. OVER_RUN;
171 2       ELSE IF (STATUS AND 20H)<0
172 2           THEN ERR_MESSAGE_PTR=. FRAME;
173 2       USART_STATUS=(USART_CMD OR 10H); /* Reset error flags on USART */
174 2       ERR_MESSAGE_INDEX = 0;
175 2       ERROR_FLAG=1;
176 2       EX1=1; /* Turn on Tx Interrupt */
177 1   END ERROR;
178 2   LINK_DISC: PROCEDURE ;
179 2       /* This procedure sends the message 'Unable to Get Online' to the terminal */
180 2       DECLARE MESSAGE_PTR WORD   AUXILIARY,
181 2       MESSAGE   BASED MESSAGE_PTR(1) BYTE   CONSTANT,
182 2       J   BYTE   AUXILIARY,
183 2       EX1_STORE   BIT;
184 3       EX1_STORE=EX1; /* Shut off async transmit interrupt */
185 2       EX1=0;
186 2       MESSAGE_PTR=. LINK;
187 2       J=0;
188 3       DO WHILE (MESSAGE(J)<0);

```



```

185 4      DO WHILE (USART_STATUS AND 01H)=0; /* Wait for TxRDY on USART */
186 4      END;
187 3      USART_DATA=MESSAGE(J);
188 3      J=J+1;
189 3      END;
190 2      EX1=EX1_STORE; /* Restore async transmit interrupt */
191 1      END LINK_DISC;

192 2      CO: PROCEDURE (CHAR) USING 2;
193 2      DECLARE CHAR    BYTE;
194 3      DO WHILE (USART_STATUS AND 01H) = 0;
195 3      END;
196 2      USART_DATA=CHAR;
197 1      END CO;
198 2      CI: PROCEDURE BYTE USING 2;
199 3      DO WHILE (USART_STATUS AND 02H) = 0;
200 3      END;
201 2      RETURN USART_DATA;
202 1      END CI;

203 2      GET_HEX: PROCEDURE BYTE USING 2;
204 2      DECLARE CHAR    BYTE    AUXILIARY,
                I            BYTE    AUXILIARY;
205 2      LO: CHAR=CI;
206 3      DO I=0 TO 15;
207 3      IF CHAR=HEX_TABLE(I)
                THEN GOTO L1;
209 3      END;
210 2      L1: CALL CO(HEX_TABLE(I));
211 2      IF I=16
                THEN GOTO LO;
213 2      RETURN I;
214 1      END GET_HEX;
215 2      OUTPUT_MESSAGE: PROCEDURE (MESSAGE_PTR) USING 2;
216 2      DECLARE MESSAGE_PTR WORD,
                MESSAGE    BASED MESSAGE_PTR(1) BYTE CONSTANT,
                I            BYTE    AUXILIARY;
217 2      I=0;
218 3      DO WHILE MESSAGE(I) <> 0;
219 3      CALL CO(MESSAGE(I));
220 3      I=I+1;

```

int f

```

221 3      END;
222 1      END OUTPUT_MESSAGE;

223 2      MENU: PROCEDURE USING 2;

224 2      DECLARE I          BYTE    AUXILIARY,
                CHAR        BYTE    AUXILIARY,
                STATION_ADDRESS BYTE    AUXILIARY;

225 2      START:
                CALL OUTPUT_MESSAGE(.SIGN_ON);

226 2      MO: CHAR=CI; /* Read a character */

227 3      DO I=0 TO 4;
228 3      IF CHAR=MENU_CHAR(I)
                THEN GOTO M1;
230 3      END;

231 2      M1: CALL CO(MENU_CHAR(I));
232 2      IF I=5
                THEN GOTO MO;

234 3      DO CASE I;

235 4      DO;
236 4      CALL OUTPUT_MESSAGE(.STAT_ADDR);
237 4      STATION_ADDRESS=SHL(GET_HEX,4);
238 4      STATION_ADDRESS=(STATION_ADDRESS OR GET_HEX);
239 4      STAD=STATION_ADDRESS;
240 4      CALL OUTPUT_MESSAGE(.S_ADDR_ACK);
241 4      CALL CO(HEX_TABLE(SHR(STATION_ADDRESS,4)));
242 4      CALL CO(HEX_TABLE(OFH AND STATION_ADDRESS));
243 4      CALL OUTPUT_MESSAGE(.ADDR_ACK_FIN);
244 4      END;

245 4      DO;

246 4      CALL OUTPUT_MESSAGE(.DEST_ADDR);
247 4      DESTINATION_ADDRESS=SHL(GET_HEX,4);
248 4      DESTINATION_ADDRESS=(DESTINATION_ADDRESS OR GET_HEX);
249 4      CALL OUTPUT_MESSAGE(.D_ADDR_ACK);

```

```

250 4          CALL CO(HEX_TABLE(SHR(DESTINATION_ADDRESS,4)));
251 4          CALL CO(HEX_TABLE(OFH AND DESTINATION_ADDRESS));
252 4          CALL OUTPUT_MESSAGE(.ADDR_ACK_FIN);
253 4          END;
254 4          DO;
255 4              CALL OUTPUT_MESSAGE(.FIN);
256 4              CALL OPEN;
257 4          END;
258 4          DO;
259 4              CALL OUTPUT_MESSAGE(.FIN);
260 4              CALL CLOSE;
261 4          END;
262 3          CALL OUTPUT_MESSAGE(.FIN);
263 3          END; /* DO CASE */
264 1          END MENU;
265 2          USART_RECV_INT: PROCEDURE INTERRUPT 0 USING 2;
266 2          DECLARE CHAR      BYTE      AUXILIARY,
                STATUS      BYTE      AUXILIARY;
267 2          CHAR=USART_DATA;
268 2          STATUS=USART_STATUS AND 3BH;
269 2          IF STATUS<>0
                THEN CALL ERROR(STATUS);
271 2          ELSE IF CHAR=ESC
                THEN CALL MENU;
273 3          ELSE DO;
274 3              CALL FIFO_T_IN(CHAR);
275 3              IF ECHO=0
                THEN CALL CO(CHAR);
277 3          END;
278 1          END USART_RECV_INT;
279 1          BEGIN;
                CALL POWER_ON;
280 2          DO FOREVER;
281 2              IF SEND_DATA<0
                THEN DO;
283 4                  DO WHILE NOT(XMIT_BUFFER_EMPTY); /*Wait until SIU_XMIT_BUFFER
                                                             is empty */
284 4                  END;
285 3                  LENGTH, CHAR =1;
286 3                  SIU_XMIT_BUFFER(0)=DESTINATION_ADDRESS;
287 4                  DO WHILE ((CHAR<>LF) AND (LENGTH<BUFFER_LENGTH) AND (BUFFER_STATUS_T<>EMPTY));

```

```

288 4      CHAR=FIFO_T_OUT;
289 4      SIU_XMIT_BUFFER(LENGTH)=CHAR;
290 4      LENGTH=LENGTH+1;
291 4      END;

/* If the line entered at the terminal is greater than BUFFER_LENGTH char, send the
first BUFFER_LENGTH char, then send the rest; since the SIU buffer is only BUFFER_LENGTH bytes */
292 3      L1:      I=0; /* Use 1 to count the number of unsuccessful
transmits */

293 3      RETRY:      RESULT=TRANSMIT(LENGTH); /* Send the message */
294 3      IF RESULT<>DATA_TRANSMITTED
THEN DO;
/* Wait 50 msec for link to connect then try again */
296 4      WAIT=1;
297 4      TH0=3CH;
298 4      TLO=0AFH;
299 4      TR0=1;
300 5      DO WHILE WAIT;
301 5      END;
302 4      TR0=0;
303 4      I=I+1;
304 5      IF I>100 THEN DO; /* Wait 5 sec to get on line else
send error message to terminal
and try again */
CALL LINK_DISC;
GOTO L1;
END;
GOTO RETRY;
END;
END;

312 2      END;
313 1      END USER*MOD;

```

WARNINGS:  
2 IS THE HIGHEST USED INTERRUPT

MODULE INFORMATION:  
CODE SIZE  
CONSTANT SIZE  
DIRECT VARIABLE SIZE  
INDIRECT VARIABLE SIZE  
BIT SIZE  
BIT-ADDRESSABLE SIZE  
AUXILIARY VARIABLE SIZE  
MAXIMUM STACK SIZE  
REGISTER-BANK(S) USED:  
713 LINES READ  
0 PROGRAM ERROR(S)  
END OF PL/M-51 COMPILATION

(STATIC+OVERLAYABLE)  
= 0682H 1714D  
= 01CFH 463D  
= 00H+05H 0D+ 5D  
= 00H+00H 0D+ 0D  
= 02H+01H 2D+ 1D  
= 00H+00H 0D+ 0D  
= 021FH 543D  
= 0028H 40D  
0 1 2



ISIS-II PL/M-51 V1.0

COMPILER INVOKED BY: :F2:PLM51 :F2:PNOTE.SRC

```

*TITLE ('RUP1-44 Primary Station')
*DEBUG
*REGISTERBANK(0)
1 1 MAIN*MOD: DO,

/* To save paper the RUP1 registers are not listed, but this is the statement
   used to include them: *INCLUDE (:F2:REQ44.DCL) */

*NDLIST

5 1 DECLARE LIT LITERALLY 'LITERALLY',
      TRUE LIT 'OFFH',
      FALSE LIT '00H',
      FOREVER LIT 'WHILE 1',

/* SDLC COMMANDS AND RESPONSES */

6 1 DECLARE SNRM LIT '93H',
      UA LIT '73H',
      DISC LIT '53H',
      DM LIT '1FH',
      FRMR LIT '97H',
      REQ_DISC LIT '53H',
      UP LIT '33H',
      TEST LIT '0F3H',
      RR LIT '11H',
      RNR LIT '15H',

/* REMOTE STATION BUFFER STATUS */

BUFFER_READY LIT '0',
BUFFER_NOT_READY LIT '1',

/* STATION STATES */
DISCONNECT_S LIT '00H', /* LOGICALLY DISCONNECTED STATE*/
GO_TO_DISC LIT '01H',
I_T_S LIT '02H', /* INFORMATION TRANSFER STATE */

/* PARAMETERS PASSED TO XMIT_I_T_S */
T_I_FRAME LIT '00H',
T_RR LIT '01H',
T_RNR LIT '02H',

/* SECONDARY STATION IDENTIFICATION */
NUMBER_OF_STATIONS LIT '2',
SECONDARY_ADDRESSES(NUMBER_OF_STATIONS)
      BYTE CONSTANT(55H, 43H),

```

```

/* Remote Station Database */
RSD(NUMBER_OF_STATIONS) STRUCTURE
(STATION_ADDRESS BYTE,
 STATION_STATE BYTE,
 NS BYTE,
 NR BYTE,
 BUFFER_STATUS BYTE, /* The status of the secondary stations buffer */
 INFO_LENGTH BYTE,
 DATA(64) BYTE) AUXILIARY,

/* VARIABLES */
STATION_NUMBER CC BYTE AUXILIARY,
RCV_FIELD_LENGTH CC BYTE AUXILIARY,
WAIT BIT,

/* BUFFERS */
SIU_XMIT_BUFFER(64) BYTE IDATA,
SIU_RECV_BUFFER(64) BYTE,

7 2 POWER_ON: PROCEDURE ;
8 2 DECLARE I BYTE AUXILIARY;
9 2 TBS= SIU_XMIT_BUFFER(0);
10 2 RBS= SIU_RECV_BUFFER(0);
11 2 RBL=64; /* 64 Byte receive buffer */
12 2 RBE=1; /* Enable the SIU's receiver */
13 3 DO I= 0 TO NUMBER_OF_STATIONS-1;
14 3 RSD(I).STATION_ADDRESS=SECONDARY_ADDRESSES(I);
15 3 RSD(I).STATION_STATE=DISCONNECT_S;
16 3 RSD(I).BUFFER_STATUS=BUFFER_NOT_READY;
17 3 RSD(I).INFO_LENGTH=0;
18 3 END;
19 2 SMD=54H; /* Using DPLL, NRZI, PFS, TIMER 1, @ 62.5 Kbps */
20 2 TMD=21H;
21 2 TH1=OFFH;
22 2 TCON=40H; /* Use timer 0 for receive time out interrupt */
23 2 IE=82H;
24 1 END POWER_ON;
25 2 XMIT: PROCEDURE (CONTROL_BYTE);
26 2 DECLARE CONTROL_BYTE BYTE;
27 2 TCB=CONTROL_BYTE;
28 2 TBF=1;

```

```

29 2      RTS=1;
30 3      DO WHILE NOT SI;
31 3      END;
32 2      SI=0;

33 1      END XMIT;

34 2      TIMER_O_INT: PROCEDURE INTERRUPT 1 USING 1;
35 2      WAIT=0;
36 1      END TIMER_O_INT;

37 2      TIME_OUT: PROCEDURE BYTE; /* Time_out returns true if there wasn't
                                   a frame received within 200 msec
                                   If there was a frame received within
                                   200 msec then time_out returns false. */

38 2      DECLARE I BYTE AUXILIARY;
39 3      DO I=0 TO 3;
40 3      WAIT=1;
41 3      TH0=3CH;
42 3      TLO=0AFH;
43 3      TRO=1;
44 4      DO WHILE WAIT;
45 4      IF SI=1 THEN GOTO T_O1;
46 4      END;
47 4      END;
48 3      END;
49 2      RETURN TRUE;

50 2      T_O1: SI=0;
51 2      RETURN FALSE;

52 1      END TIME_OUT;

53 2      SEND_DISC: PROCEDURE;
54 2      TBL=0;
55 2      CALL XMIT(DISC);
56 2      IF TIME_OUT=FALSE THEN IF RCB=UA OR RCB=DM THEN DO;
57 3      THEN IF RCB=UA OR RCB=DM THEN DO;
58 3      RSD(STATION_NUMBER).BUFFER_STATUS=BUFFER_NOT_READY;
59 3      RSD(STATION_NUMBER).STATION_STATE=DISCONNECT_S;
60 3      END;
61 3      RBE=1;
62 2      END SEND_DISC;

63 1      END SEND_DISC;

64 2      SEND_SNRM: PROCEDURE;
65 2      TBL=0;

```

```

66 2      CALL XMIT(SNRM);
67 2      IF (TIME_OUT=FALSE) AND (RCB=UA)
          THEN DO;
69 3          RSD(STATION_NUMBER).STATION_STATE=I_T_S;
70 3          RSD(STATION_NUMBER).NS=0;
71 3          RSD(STATION_NUMBER).NR=0;
72 3      END;
73 2      RBE=1;

74 1      END SEND_SNRM;

75 2      CHECK_NS: PROCEDURE BYTE;

          /* Check the Ns Field of the received frame. If Nr(P)=Ns(S) return true */
76 2      IF (RSD(STATION_NUMBER).NR=(SHR(RCB,1) AND 07H))
          THEN RETURN TRUE;
78 2      ELSE RETURN FALSE;

79 1      END CHECK_NS;

80 2      CHECK_NR: PROCEDURE BYTE;

          /* Check the Nr field of the received frame. If Ns(P)+1=Nr(S) then the frame
          has been acknowledged, else if Ns(P)=Nr(S) then the frame has not been
          acknowledged, else reset the secondary */
81 2      IF (((RSD(STATION_NUMBER).NS + 1) AND 07H) = SHR(RCB,5))
          THEN DO;
83 3          RSD(STATION_NUMBER).NS=((RSD(STATION_NUMBER).NS+1) AND 07H);
84 3          RSD(STATION_NUMBER).INFO_LENGTH=0;
85 3      END;
86 2      ELSE IF (RSD(STATION_NUMBER).NS <> SHR(RCB,5))
          THEN RETURN FALSE;

88 2      RETURN TRUE;

89 1      END CHECK_NR;

90 2      RECEIVE: PROCEDURE

91 2      DECLARE I BYTE AUXILIARY;

92 2      RSD(STATION_NUMBER).BUFFER_STATUS=BUFFER_READY;

          /* If an RNR was received buffer_status will be changed in the supervisory
          frame decode section futher down in this procedure, any other response
          means the remote stations buffer is ready */
93 2      IF (RCB AND 01H)=0
          THEN DO; /* I Frame Received */
95 3          IF (CHECK_NS=TRUE AND BOV=0 AND CHECK_NR=TRUE)
          THEN DO;
97 4              RSD(STATION_NUMBER).NR=((RSD(STATION_NUMBER).NR+1) AND 07H);
98 4              RBP=1;

```



```

99 4          RECV_FIELD_LENGTH=RFL-1;
100 4          END;
          ELSE RSD(STATION_NUMBER).STATION_STATE=GO_TO_DISC;
101 3          END;
102 3          ELSE IF (RCB AND 03H)=01H
103 2          THEN DO; /* Supervisory frame received */
105 3              IF CHECK_NR=FALSE
                  THEN RSD(STATION_NUMBER).STATION_STATE=GO_TO_DISC;
107 3              ELSE IF ((RCB AND 0FH)=05H) /* then RNR */
                  THEN RSD(STATION_NUMBER).BUFFER_STATUS=BUFFER_NOT_READY;
109 3              END;
          ELSE DO; /* Unnumbered frame or unknown frame received */
110 3              IF RCB=FRMR
111 3              THEN DO; /* If FRMR was received check Nr for an
                          acknowledged I frame */
113 4                  RCB=SIU_RECV_BUFFER(1);
114 4                  I=CHECK_NR;
115 4                  END;
116 3                  RSD(STATION_NUMBER).STATION_STATE=GO_TO_DISC;
117 3                  END;
118 2          RBE=1;
119 1          END RECEIVE;

120 2          XMIT_I_T_S: PROCEDURE (TEMP);
121 2          DECLARE    TEMP    BYTE;
122 2          IF TEMP=T_I_FRAME
          THEN DO; /* Transmit I frame */
                  /* Transfer the station buffer into internal ram */
124 4                  DO TEMP=0 TO RSD(STATION_NUMBER).INFO_LENGTH-1;
125 4                  SIU_XMIT_BUFFER(TEMP)=RSD(STATION_NUMBER).DATA(TEMP);
126 4                  END;
                  /* Build the I frame control field */
127 3                  TEMP=(SHL(RSD(STATION_NUMBER).NR,5) OR SHL(RSD(STATION_NUMBER).NS,1) OR 10H);
128 3                  TBL=RSD(STATION_NUMBER).INFO_LENGTH;
129 3                  CALL XMIT(TEMP);
130 3                  IF TIME_OUT=FALSE
                  THEN CALL RECEIVE;
132 3                  END;
          ELSE DO; /* Transmit RR or RNR */
133 3                  IF TEMP=T_RR
134 3                  THEN TEMP=RR;
                  ELSE TEMP=RNR;
136 3                  END;

```

```

137 3          TEMP:=(SHL(RSD(STATION_NUMBER).NR,5) OR TEMP);
138 3          TBL=0;
139 3          CALL XMIT(TEMP);
140 3          IF TIME_OUT=FALSE
141             THEN CALL RECEIVE;
142 3          END;
143 1          END XMIT_I_T_S;
144 2          BUFFER_TRANSFER: PROCEDURE;
145 2          DECLARE      I      BYTE    AUXILIARY;
146                        J      BYTE    AUXILIARY;
147 3          DO I=0 TO NUMBER_OF_STATIONS-1;
148             IF RSD(I).STATION_ADDRESS=SIU_RECV_BUFFER(0)
149                THEN GOTO T1;
150 2          T1:  IF I=NUMBER_OF_STATIONS /* If the addressed station does not exists,
151                        then discard the data */
152             THEN DO;
153                 RBP=0;
154                 RETURN;
155             END;
156             ELSE IF RSD(I).INFO_LENGTH=0
157                 THEN DO;
158                     RSD(I).INFO_LENGTH=RCV_FIELD_LENGTH;
159                     DO J=1 TO RCV_FIELD_LENGTH;
160                         RSD(I).DATA(J-1)=SIU_RECV_BUFFER(J);
161                     END;
162                     RBP=0;
163             END;
164 1          END BUFFER_TRANSFER;
165 1          BEGIN;
166             CALL POWER_ON;
167 2          DO FOREVER;
168 3          DO STATION_NUMBER=0 TO NUMBER_OF_STATIONS-1;
169             STAD=RSD(STATION_NUMBER).STATION_ADDRESS;
170             IF RSD(STATION_NUMBER).STATION_STATE = DISCONNECT_S
171                 THEN CALL SEND_SNRH;
172             ELSE IF RSD(STATION_NUMBER).STATION_STATE = GO_TO_DISC
173                 THEN CALL SEND_DISC;
174             ELSE IF ((RSD(STATION_NUMBER).INFO_LENGTH>0) AND
175                     (RSD(STATION_NUMBER).BUFFER_STATUS=BUFFER_READY))
176                 THEN CALL XMIT_I_T_S(T_I_FRAME);
177             ELSE IF RBP=0
178                 THEN CALL XMIT_I_T_S(T_RR);
179             ELSE CALL XMIT_I_T_S(T_RNR);
180             IF RBP=1
181                 THEN CALL BUFFER_TRANSFER;

```

```

179 3          END;
180 2          END;
181 1          END MAIN#MOD;

```

WARNINGS:  
1 IS THE HIGHEST USED INTERRUPT

MODULE INFORMATION:	(STATIC+OVERLAYABLE)
CODE SIZE	= 053DH 1341D
CONSTANT SIZE	= 0002H 2D
DIRECT VARIABLE SIZE	= 40H+02H 64D+ 2D
INDIRECT VARIABLE SIZE	= 40H+00H 64D+ 0D
BIT SIZE	= 01H+00H 1D+ 0D
BIT-ADDRESSABLE SIZE	= 00H+00H 0D+ 0D
AUXILIARY VARIABLE SIZE	= 0093H 147D
MAXIMUM STACK SIZE	= 0019H 25D
REGISTER-BANK(S) USED:	0 1
456 LINES READ	
0 PROGRAM ERROR(S)	
END OF PL/M-51 COMPILATION	









# 

## 

### 

#### 

##### 

###### 

###### 

###### 

###### 

###### 

###### 

###### 

###### 

###### 

###### 

###### 

###### 

###### 

###### 

###### 

###### 

###### 

###### 

###### 

###### 

###### 

###### 

###### 

######

Table 1. RUPITM -44 Family Pin Description

**VSS**  
Circuit ground potential.

**VCC**  
+5V power supply during operation and program verification.

**PORT 0**

Port 0 is an 8-bit open drain bidirectional I/O port. It is also the multiplexed low-order address and data bus when using external memory. It is used for data output during program verification. Port 0 can sink/source eight LS TTL loads.

**PORT 1**

Port 1 is an 8-bit quasi-bidirectional I/O port. It is used for the low-order address byte during program verification. Port 1 can sink/source four LS TTL loads.

In non-loop mode two of the I/O lines serve alternate functions:

- RTS (P1.6). Request-to-Send output. A low indicates that the RUPITM-44 is ready to transmit.
- CTS (P1.7) Clear-to-Send input. A low indicates that a receiving station is ready to receive.

**PORT 2**

Port 2 is an 8-bit quasi-bidirectional I/O port. It also emits the high-order address byte when accessing external memory. It is used for the high-order address and the control signals during program verification. Port 2 can sink/source four LS TTL loads.

**PORT 3**

Port 3 is an 8-bit quasi-bidirectional I/O port. It also contains the interrupt, timer, serial port and RD and WR pins that are used by various options. The output latch corresponding to a secondary function must be programmed to a one (1) for that function to operate. Port 3 can sink/source four LS TTL loads.

In addition to I/O, some of the pins also serve alternate functions as follows:

- I/O RxD (P3.0). In point-to-point or multipoint configurations, this pin controls the direction of pin P3.1. Serves as Receive Data input in loop and diagnostic modes.
- DATA TxD (P3.1) In point-to-point or multipoint configurations, this pin functions as data input/output. In loop mode, it serves as transmit pin. A '0' written to this pin enables diagnostic mode.
- INT0 (P3.2). Interrupt 0 input or gate control input for counter 0.
- INT1 (P3.3). Interrupt 1 input or gate control input for counter 1.
- TO(P3.4). Input to counter 0.

- SCLK T1 (P3.5). In addition to I/O, this pin provides input to counter 1 or serves as SCLK (serial clock) input.
- WR (P3.6). The write control signal latches the data byte from Port 0 into the External Data Memory.
- RD (P3.7). The read control signal enables External Data Memory to Port 0.

**RST**

A high on this pin for two machine cycles while the oscillator is running resets the device. A small external pulldown resistor ( $\approx 8.2K\Omega$ ) from RST to VSS permits power-on reset when a capacitor ( $\approx 10\mu f$ ) is also connected from this pin to VCC.

**ALE/PROG**

Provides Address Latch Enable output used for latching the address into external memory during normal operation. It is activated every six oscillator periods except during an external data memory access. It also receives the program pulse input for programming the EPROM version.

**PSEN**

The Program Store Enable output is a control signal that enables the external Program Memory to the bus during external fetch operations. It is activated every six oscillator periods, except during external data memory accesses. Remains high during internal program execution.

**EA/VPP**

When held at a TTL high level, the RUPITM-44 executes instructions from the internal ROM when the PC is less than 4096. When held at a TTL low level, the RUPITM-44 fetches all instructions from external Program Memory. The pin also receives the 21V EPROM programming supply voltage on the 8744.

**XTAL 1**

Input to the oscillator's high gain amplifier. Required when a crystal is used. Connect to VSS when external source is used on XTAL 2.

**XTAL 2**

Output from the oscillator's amplifier. Input to the internal timing circuitry. A crystal or external source can be used.

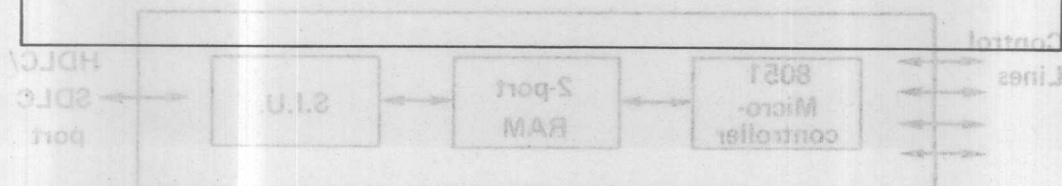


Figure 1. Dual Controller Architecture

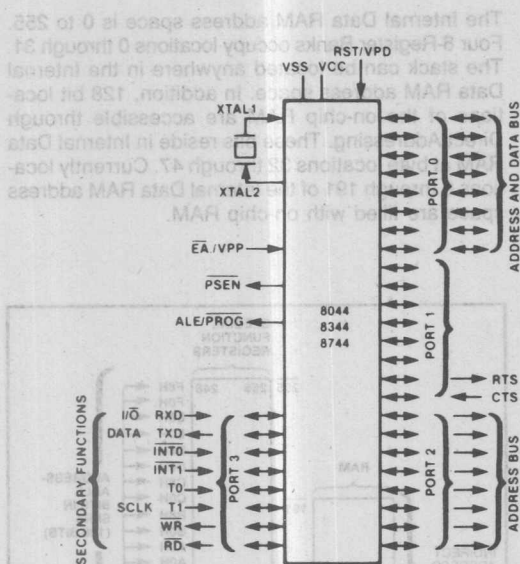


Figure 2.  
Logic Symbol

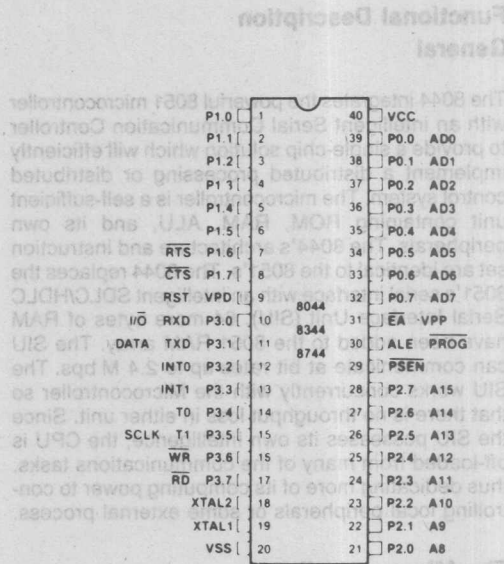


Figure 3. Pin  
Configuration

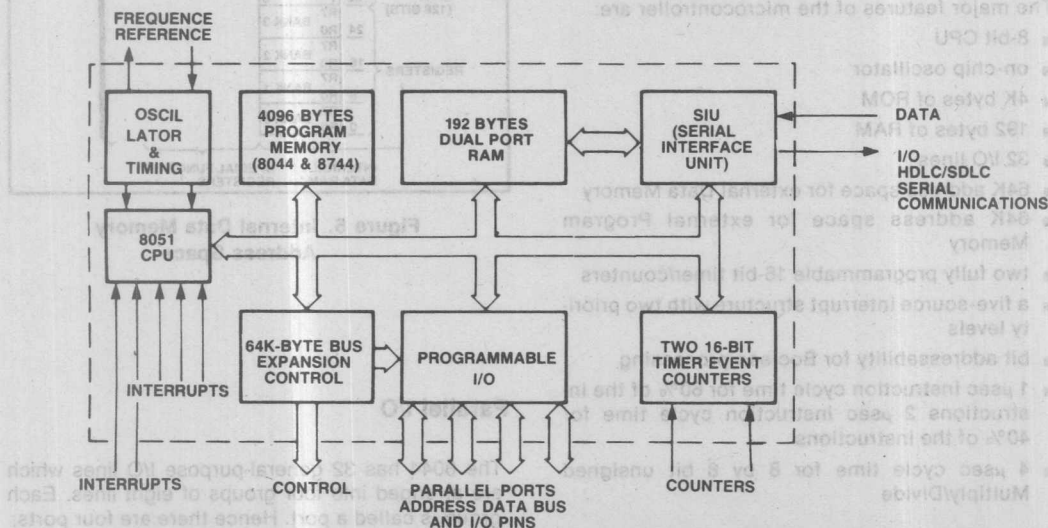


Figure 4.  
Block Diagram



## Functional Description

### General

The 8044 integrates the powerful 8051 microcontroller with an intelligent Serial Communication Controller to provide a single-chip solution which will efficiently implement a distributed processing or distributed control system. The microcontroller is a self-sufficient unit containing ROM, RAM, ALU, and its own peripherals. The 8044's architecture and instruction set are identical to the 8051's. The 8044 replaces the 8051's serial interface with an intelligent SDLC/HDLC Serial Interface Unit (SIU). 64 more bytes of RAM have been added to the 8051 RAM array. The SIU can communicate at bit rates up to 2.4 M bps. The SIU works concurrently with the Microcontroller so that there is no throughput loss in either unit. Since the SIU possesses its own intelligence, the CPU is off-loaded from many of the communications tasks, thus dedicating more of its computing power to controlling local peripherals or some external process.

### The Microcontroller

The microcontroller is a stand-alone high-performance single-chip computer intended for use in sophisticated real-time application such as instrumentation, industrial control, and intelligent computer peripherals.

The major features of the microcontroller are:

- 8-bit CPU
- on-chip oscillator
- 4K bytes of ROM
- 192 bytes of RAM
- 32 I/O lines
- 64K address space for external Data Memory
- 64K address space for external Program Memory
- two fully programmable 16-bit timer/counters
- a five-source interrupt structure with two priority levels
- bit addressability for Boolean processing
- 1  $\mu$ sec instruction cycle time for 60% of the instructions
- 2  $\mu$ sec instruction cycle time for 40% of the instructions
- 4  $\mu$ sec cycle time for 8 by 8 bit unsigned Multiply/Divide

### Internal Data Memory

Functionally the Internal Data Memory is the most flexible of the address spaces. The Internal Data Memory space is subdivided into a 256-byte Internal Data RAM address space and a 128-byte Special Function Register address space as shown in Figure 5.

The Internal Data RAM address space is 0 to 255. Four 8-Register Banks occupy locations 0 through 31. The stack can be located anywhere in the Internal Data RAM address space. In addition, 128 bit locations of the on-chip RAM are accessible through Direct Addressing. These bits reside in Internal Data RAM at byte locations 32 through 47. Currently locations 0 through 191 of the Internal Data RAM address space are filled with on-chip RAM.

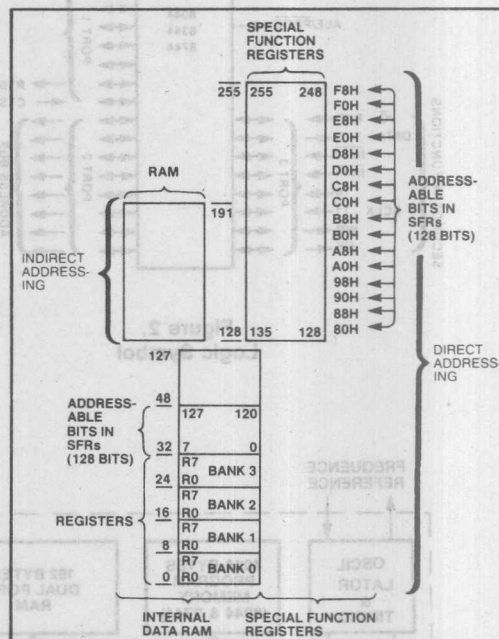


Figure 5. Internal Data Memory Address Space

### Parallel I/O

The 8044 has 32 general-purpose I/O lines which are arranged into four groups of eight lines. Each group is called a port. Hence there are four ports; Port 0, Port 1, Port 2, and Port 3. Up to five lines from 1 and Port 2 are dedicated to supporting the serial channel when the SIU is invoked. Due to the nature of the serial port, two of Port 3's I/O lines (P3.0 and P3.1) do not have latched outputs. This is true whether or not the serial channel is used.

Table 1. MCS®-51 Instruction Set Description

ARITHMETIC OPERATIONS				LOGICAL OPERATIONS (CONTINUED)			
Mnemonic		Description	Byte Cyclic	Mnemonic	Destination		Byte Cyclic
ADD	A,Rn	Add register to Accumulator	1 1	ORL	A,@Ri	OR indirect RAM to Accumulator	1 1
ADD	A,direct	Add direct byte to Accumulator	2 1	ORL	A,#data	OR immediate data to Accumulator	2 1
ADD	A,@Ri	Add indirect RAM to Accumulator	1 1	ORL	direct,A	OR Accumulator to direct byte	2 1
ADD	A,#data	Add immediate data to Accumulator	2 1	ORL	direct,#data	OR immediate data to direct byte	3 2
ADDC	A,Rn	Add register to Accumulator with Carry	1 1	XRL	A,Rn	Exclusive-OR register to Accumulator	1 1
ADDC	A,direct	Add direct byte to A with Carry flag	2 1	XRL	A,direct	Exclusive-OR direct byte to Accumulator	2 1
ADDC	A,@Ri	Add indirect RAM to A with Carry flag	1 1	XRL	A,@Ri	Exclusive-OR indirect RAM to A	1 1
ADDC	A,#data	Add immediate data to A with Carry flag	2 1	XRL	A,#data	Exclusive-OR immediate data to A	2 1
SUBB	A,Rn	Subtract register from A with Borrow	1 1	XRL	direct,A	Exclusive-OR Accumulator to direct byte	2 1
SUBB	A,direct	Subtract direct byte from A with Borrow	2 1	XRL	direct,#data	Exclusive-OR immediate data to direct byte	3 2
SUBB	A,@Ri	Subtract indirect RAM from A with Borrow	1 1	CLR	A	Clear Accumulator	1 1
SUBB	A,#data	Subtract immediate data from A with Borrow	2 1	CPL	A	Complement Accumulator	1 1
INC	A	Increment Accumulator	1 1	RL	A	Rotate Accumulator Left	1 1
INC	Rn	Increment register	1 1	RLC	A	Rotate A Left through the Carry flag	1 1
INC	direct	Increment direct byte	2 1	RR	A	Rotate Accumulator Right	1 1
INC	@Ri	Increment indirect RAM	1 1	RRC	A	Rotate A Right through Carry flag	1 1
INC	DPTR	Increment Data Pointer	1 2	SWAP	A	Swap nibbles within the Accumulator	1 1
DEC	A	Decrement Accumulator	1 1				
DEC	Rn	Decrement register	1 1				
DEC	direct	Decrement direct byte	2 1				
DEC	@Ri	Decrement indirect RAM	1 1				
MUL	AB	Multiply A & B	1 4				
DIV	AB	Divide A by B	1 4				
DA	A	Decimal Adjust Accumulator	1 1				
LOGICAL OPERATIONS				DATA TRANSFER			
Mnemonic		Destination	Byte Cyclic	Mnemonic	Description		Byte Cyclic
ANL	A,Rn	AND register to Accumulator	1 1	MOV	A,Rn	Move register to Accumulator	1 1
ANL	A,direct	AND direct byte to Accumulator	2 1	MOV	A,direct	Move direct byte to Accumulator	2 1
ANL	A,@Ri	AND indirect RAM to Accumulator	1 1	MOV	A,@Ri	Move indirect RAM to Accumulator	1 1
ANL	A,#data	AND immediate data to Accumulator	2 1	MOV	A,#data	Move immediate data to Accumulator	2 1
ANL	direct,A	AND Accumulator to direct byte	2 1	MOV	Rn,A	Move Accumulator to register	1 1
ANL	direct,#data	AND immediate data to direct byte	3 2	MOV	Rn,direct	Move direct byte to register	2 2
ORL	A,Rn	OR register to Accumulator	1 1	MOV	Rn,#data	Move immediate data to register	2 1
ORL	A,direct	OR direct byte to Accumulator	2 1	MOV	direct,A	Move Accumulator to direct byte	2 1
				MOV	direct,Rn	Move register to direct byte	2 2
				MOV	direct,direct	Move direct byte to direct	3 2
				MOV	direct,@Ri	Move indirect RAM to direct byte	2 2

Table 1. (Cont.)

DATA TRANSFER (CONTINUED)				PROGRAM AND MACHINE CONTROL			
Mnemonic		Description	Byte Cyc	Mnemonic		Description	Byte Cyc
MOV	direct,#data	Move immediate data to direct byte	3 2	ACALL	addr11	Absolute Subroutine Call	2 2
MOV	@Ri,A	Move Accumulator to indirect RAM	1 1	LCALL	addr16	Long Subroutine Call	3 2
MOV	@Ri,direct	Move direct byte to indirect RAM	2 2	RET		Return from subroutine	1 2
MOV	@Ri,#data	Move immediate data to indirect RAM	2 1	RETI		Return from interrupt	1 2
MOV	DPTR,#data16	Load Data Pointer with a 16-bit constant	3 2	AJMP	addr11	Absolute Jump	2 2
MOVC	A,@A+DPTR	Move Code byte relative to DPTR to A	1 2	LJMP	addr16	Long Jump	3 2
MOVC	A,@A+PC	Move Code byte relative to PC to A	1 2	SJMP	rel	Short Jump (relative addr)	2 2
MOVX	A,@Ri	Move External RAM (8-bit addr) to A	1 2	JMP	@A+DPTR	Jump indirect relative to the DPTR	1 2
MOVX	A,@DPTR	Move External RAM (16-bit addr) to A	1 2	JZ	rel	Jump if Accumulator is Zero	2 2
MOVX	@Ri,A	Move A to External RAM (8-bit addr)	1 2	JNZ	rel	Jump if Accumulator is Not Zero	2 2
MOVX	@DPTR,A	Move A to External RAM (16-bit addr)	1 2	JC	rel	Jump if Carry flag is set	2 2
PUSH	direct	Push direct byte onto stack	2 2	JNC	rel	Jump if No Carry flag	2 2
POP	direct	Pop direct byte from stack	2 2	JB	bit,rel	Jump if direct Bit set	3 2
XCH	A,Rn	Exchange register with Accumulator	1 1	JNB	bit,rel	Jump if direct Bit Not set	3 2
XCH	A,direct	Exchange direct byte with Accumulator	2 1	JBC	bit,rel	Jump if direct Bit is set & Clear bit	3 2
XCH	A,@Ri	Exchange indirect RAM with A	1 1	CJNE	A,direct,rel	Compare direct to A & Jump if Not Equal	3 2
XCHD	A,@Ri	Exchange low-order Digit ind RAM w A	1 1	CJNE	A,#data,rel	Comp, immed, to A & Jump if Not Equal	3 2
BOOLEAN VARIABLE MANIPULATION				CJNE	Rn,#data,rel	Comp, immed, to reg & Jump if Not Equal	3 2
Mnemonic		Description	Byte Cyc	CJNE	@Ri,#data,rel	Comp, immed, to ind, & Jump if Not Equal	3 2
CLR	C	Clear Carry flag	1 1	DJNZ	Rn,rel	Decrement register & Jump if Not Zero	2 2
CLR	bit	Clear direct bit	2 1	DJNZ	direct,rel	Decrement direct & Jump if Not Zero	3 2
SETB	C	Set Carry flag	1 1	NOP		No operation	1 1
SETB	bit	Set direct Bit	2 1	<b>Notes on data addressing modes:</b>			
CPL	C	Complement Carry flag	1 1	Rn		Working register R0-R7	
CPL	bit	Complement direct bit	2 1	direct		128 internal RAM locations, any I/O port, control or status register	
ANL	C,bit	AND direct bit to Carry flag	2 2	@Ri		Indirect internal RAM location addressed by register R0 or R1	
ANL	C,/bit	AND complement of direct bit to Carry	2 2	#data		8-bit constant included in instruction	
ORL	C,bit	OR direct bit to Carry flag	2 2	#data16		16-bit constant included as bytes 2 & 3 of instruction	
ORL	C,/bit	OR complement of direct bit to Carry	2 2	bit		128 software flags, any I/O pin, control or status bit	
MOV	C,/bit	Move direct bit to Carry flag	2 1	<b>Notes on program addressing modes:</b>			
MOV	bit,C	Move Carry flag to direct bit	2 2	addr16		Destination address for LCALL & LJMP may be anywhere within the 64-K program memory address space	
				Addr11		Destination address for ACALL & AJMP will be within the same 2-K page of program memory as the first byte of the following instruction	
				rel		SJMP and all conditional jumps include an 8-bit offset byte, Range is +127-128 bytes relative to first byte of the following instruction	
				All mnemonics copyrighted © Intel Corporation 1979			

Port 0 and Port 2 also have an alternate dedicated function. When placed in the external access mode, Port 0 and Port 2 become the means by which the 8044 communicates with external program memory. Port 0 and Port 2 are also the means by which the 8044 communicates with external data memory. Peripherals can be memory mapped into the address space and controlled by the 8044.

### Timer/Counters

The 8044 contains two 16-bit counters which can be used for measuring time intervals, measuring pulse widths, counting events, generating precise periodic interrupt requests, and clocking the serial communications. Internally the Timers are clocked at 1/12 of the crystal frequency, which is the instruction cycle time. Externally the counters can run up to 500 KHz.

### Interrupt System

External events and the real-time driven on-chip peripherals require service by the CPU asynchronous to the execution of any particular section of code. To tie the asynchronous activities of these functions to normal program execution, a sophisticated multiple-source, two priority level, nested interrupt system is provided. Interrupt response latency ranges from 3  $\mu$ sec to 7  $\mu$ sec when using a 12 MHz clock.

All five interrupt sources can be mapped into one of the two priority levels. Each interrupt source can be enabled or disabled individually or the entire interrupt system can be enabled or disabled. The five interrupt sources are: Serial Interface Unit, Timer 1, Timer 2, and two external interrupts. The external interrupts can be either level or edge triggered.

### Serial Interface Unit (SIU)

The Serial Interface Unit is used for HDLC/SDLC communications. It handles Zero Bit Insertion/Deletion, Flags, automatic address recognition, and a 16-bit cyclic redundancy check. In addition it implements in hardware a subset of the SDLC protocol such that it responds to many SDLC frames without CPU intervention. In certain applications it is advantageous to have the CPU control the reception or transmission of every single frame. For this reason the SIU has two modes of operation: "AUTO" and "FLEXIBLE" (or "NON-AUTO"). It is in the AUTO mode that the SIU responds to SDLC frames without CPU intervention; whereas, in the FLEXIBLE mode the reception or transmission of every single frame will be under CPU control.

There are three control registers and eight parameter registers that are used to operate the serial interface. These registers are shown in Figure 5 and Figure 6. The control registers set the modes of operation and provide status information. The eight parameter registers buffer the station address, receive and transmit control bytes, and point to the on-chip transmit and receive buffers.

Data to be received or transmitted by the SIU must be buffered anywhere within the 192 bytes of on-chip RAM. Transmit and receive buffers are not allowed to "wrap around" in RAM; a "buffer end" is generated after address 191 is reached.

### AUTO Mode

In the AUTO mode the SIU implements in hardware a subset of the SDLC protocol such that it responds to many SDLC frames without CPU intervention. All AUTO mode responses to the primary station will conform to IBM's SDLC definition. The advantages of the AUTO mode are that less software is required to implement a secondary station, and the hardware generated response to polls is much faster than doing it in software. However, the Auto mode can not be used at a primary station.

To transmit in the AUTO mode the CPU must load the Transmit Information Buffer, Transmit Buffer Start register, Transmit Buffer Length register, and set the Transmit Buffer Full bit. The SIU automatically responds to a poll by transmitting an information frame with the P/F bit in the control field set. When the SIU receives a positive acknowledgement from the primary station, it automatically increments the Ns field in the NSNR register and interrupts the CPU. A negative acknowledgement would cause the SIU to retransmit the frame.

To receive in the AUTO mode, the CPU loads the Receive Buffer Start register, the Receive Buffer Length register, clears the Receive Buffer Protect bit, and sets the Receive Buffer Empty bit. If the SIU is polled in this state, and the TBF bit indicates that the Transmit Buffer is empty, an automatic RR response will be generated. When a valid information frame is received the SIU will automatically increment Nr in the NSNR register and interrupt the CPU.

While in the AUTO mode the SIU can recognize and respond to the following commands without CPU intervention: I (Information), RR (Receive Ready), RNR (Receive Not Ready), REJ (Reject), and UP (Unnumbered Poll). The SIU can generate





the following responses without CPU intervention: I (Information), RR (Receive Ready), and RNR (Receive Not Ready).

When the Receive Buffer Empty bit (RBE) indicates that the Receive Buffer is empty, the receiver is enabled, and when the RBE bit indicates that the Receive Buffer is full, the receiver is disabled. Assuming that the Receive Buffer is empty, the SIU will respond to a poll with an I frame if the Transmit Buffer is full. If the Transmit Buffer is empty, the SIU will respond to a poll with a RR command if the Receive Buffer Protect bit (RBP) is cleared, or an RNR command if RBP is set.

### FLEXIBLE (or NON-AUTO) Mode

In the FLEXIBLE mode all communications are under control of the CPU. It is the CPU's task to encode and decode control fields, manage acknowledgements, and adhere to the requirements of the HDLC/SDLC protocols. The 8044 can be used as a primary or a secondary station in this mode.

To receive a frame in the FLEXIBLE mode, the CPU must load the Receive Buffer Start register, the Receive Buffer Length register, clear the Receive Buffer Protect bit, and set the Receive Buffer Empty bit. If a valid opening flag is received and the address field matches the byte in the Station Address register or the address field contains a broadcast address, the 8044 loads the control field in the receive control byte register, and loads the I field in the receive buffer. If there is no CRC error, the SIU interrupts the CPU, indicating a frame has just been received. If there is a CRC error, no interrupt occurs. The Receive Field Length register provides the number of bytes that were received in the information field.

To transmit a frame, the CPU must load the transmit information buffer, the Transmit Buffer Start register, the Transmit Buffer Length register, the Transmit Control Byte, and set the TBF and the RFS bit. The SIU, unsolicited by an HDLC/SDLC frame, will transmit the entire information frame, and interrupt the CPU, indicating the completion of transmission. For supervisory frames or unnumbered frames, the transmit buffer length would be 0.

### CRC

The FCS register is initially set to all 1's prior to calculating the FCS field. The SIU will not interrupt the CPU if a CRC error occurs (in both AUTO and FLEXIBLE modes). The CRC error is cleared upon receiving of an opening flag.

### Frame Format Options

In addition to the standard SDLC frame format, the 8044 will support the frames displayed in Figure 7. The standard SDLC frame is shown at the top of this figure. For the remaining frames the information field will incorporate the control or address bytes and the frame check sequences; therefore these fields will be stored in the Transmit and Receive buffers. For example, in the non-buffered mode the third byte is treated as the beginning of the information field. In the non-addressed mode, the information field begins after the opening flag. The mode bits to set the frame format options are found in the Serial Mode register and the Status register.

### EXTENDED ADDRESSING

To realize an extended control field or an extended address field using the HDLC protocol, the FLEXIBLE mode must be used. For an extended control field, the SIU is programmed to be in the non-buffered mode. The extended control field will be the first and second bytes in the Receive and Transmit Buffers. For extended addressing the SIU is placed in the non-addressed mode. In this mode the CPU must implement the address recognition for received frames. The addressing field will be the initial bytes in the Transmit and Receive buffers followed by the control field.

The SIU can transmit and receive only frames which are multiples of 8 bits. For frames received with other than 8-bit multiples, a CRC error will cause the SIU to reject the frame.

### SDLC Loop Networks

The SIU can be used in a ADLC loop as a secondary or primary station. When the SIU is placed in the Loop mode it receives the data on pin 10 and transmits the data one bit time delayed on pin 11. It can also recognize the Go ahead signal and change it into a flag when it is ready to transmit. As a secondary station the SIU can be used in the AUTO or FLEXIBLE modes. As a primary station the FLEXIBLE mode is used; however, additional hardware is required for generating the Go Ahead bit pattern. In the Loop mode the maximum data rate is 1 Mbps clocked or 375 Kbps self-clocked.

### SDLC Multidrop Networks

The SIU can be used in a SDLC non-loop configuration as a secondary or primary station. When the SIU is placed in the non-loop mode, data is received and transmitted on pin 11, and pin 10 drives a tri-state buffer. In non-loop mode, modem interface pins, RTS and CTS, become available.

FRAME OPTION	NFCS	NB	AM	FRAME FORMAT
Standard SDLC NON-AUTO Mode	0	0	0	F A C I FCS F
Standard SDLC AUTO Mode	0	0	1	F A C I FCS F
Non-Buffered Mode NON-AUTO Mode	0	1	1	F A I FCS F
Non-Buffered Mode NON-AUTO Mode	1	1	1	F A I F
No FCS Field NON-AUTO Mode	1	0	0	F A C I F
No FCS Field NON-AUTO Mode	1	0	1	F A C I F
No FCS Field Non-Buffered Mode NON-AUTO Mode	1	1	1	F A I F
No FCS Field Non-Buffered Mode NON-AUTO Mode	1	1	0	F I F

**Mode Bits:**  
 AM — "AUTO" Mode/Addressed Mode  
 NB — Non-Buffered Mode  
 NFCS — No FCS Field Mode

**Key to Abbreviations:**  
 F = Flag (01111110) I = Information Field  
 A = Address Field FCS = Frame Check Sequence  
 C = Control Field

**Note 1:** The AM bit function is controlled by the NB bit. When NB = 0, AM becomes AUTO mode select, when NB = 1, AM becomes Address mode select.

Figure 7. Frame Format Options

## Data Clocking Options

The 8044's serial port can operate in an externally clocked or self clocked system. A clocked system provides to the 8044 a clock synchronization to the data. A self-clocked system uses the 8044's on-chip Digital Phase Locked Loop (DPLL) to recover the clock from the data, and clock this data into the Serial Receive Shift Register.

In this mode, a clock synchronized with the data is externally fed into the 8044. This clock may be generated from an External Phase Locked Loop, or possibly supplied along with the data. The 8044 can transmit and receive data in this mode at rates up to 2.4 Mbps.

This self clocked mode allows data transfer without a common system data clock. An on-chip Digital Phase Locked Loop is employed to recover the data clock which is encoded in the data stream. The DPLL will converge to the nominal bit center within eight

bit transitions, worst case. The DPLL requires a reference clock of either 16 times (16x) or 32 times (32x) the data rate. This reference clock may be externally applied or internally generated. When internally generated either the 8044's internal logic clock (crystal frequency divided by two) or the timer 1 overflow is used as the reference clock. Using the internal timer 1 clock the data rates can vary from 244 to 62.5 Kbps. Using the internal logic clock at a 16x sampling rate, receive data can either be 187.5 Kbps, or 375 Kbps. When the reference clock for the DPLL is externally applied the data rates can vary from 0 to 375 Kbps at a 16x sampling rate.

To aid in a Phase Locked Loop capture, the SIU has a NRZI (Non Return to Zero Inverted) data encoding and decoding option. Additionally the SIU has a pre-frame sync option that transmits two bytes of alternating 1's and 0's to ensure that the receive station DPLL will be synchronized with the data by the time it receives the opening flag.



## Control and Status Registers

There are three SIU Control and Status Registers:

Serial Mode Register (SMD)

Status/Command Register (STS)

Send/Receive Count Register (NSNR)

The SMD, STS, and NSNR registers are all cleared by system reset. This assures that the SIU will power up in an idle state (neither receiving nor transmitting).

These registers, and their bit assignments are described below.

### SMD: Serial Mode Register (byte-addressable)

Bit: 7	6	5	4	3	2	1	0
SCM2	SCM1	SCM0	NRZI	LOOP	PFS	NB	NFCS

The Serial Mode Register (Address C9H) selects the operational modes of the SIU. The 8044 CPU can both read and write SMD. The SIU can read SMD but cannot write to it. To prevent conflict between CPU and SIU access to SMD, the CPU should write SMD only when the Request To Send (RTS) and Receive Buffer Empty (RBE) bits (in the STS register) are both false (0). Normally, SMD is accessed only during initialization.

The individual bits of the Serial Mode Register are as follows:

Bit #	Name	Description
SMD.0	NFCS	No FCS field in the SDLC frame.
SMD.1	NB	Non-Buffered mode. No control field in the SDLC frame.
SMD.2	PFS	Pre-Frame Sync mode. In this mode, the 8044 transmits two bytes before the first flag of a frame, for DPLL synchronization. If NRZI is enabled, 00H is sent; otherwise, 55H is sent. In either case, 16 pre-frame transitions are guaranteed.
SMD.3	LOOP	Loop configuration.
SMD.4	NRZI	NRZI coding option.
SMD.5	SCM0	Select Clock Mode — Bit 0
SMD.6	SCM1	Select Clock Mode — Bit 1
SMD.7	SCM2	Select Clock Mode — Bit 2

The SCM bits decode as follows:

SCM	2	1	0	Clock Mode	Data Rate (Bits/sec)*
0	0	0	0	Externally clocked	0-2.4M**
0	0	1	0	Reserved	
0	1	0	0	Self clocked, timer overflow	244-62.5K
0	1	1	0	Reserved	
1	0	0	0	Self clocked, external 16x	0-375K

SCM	2	1	0	Clock Mode	Data Rate (Bits/sec)*
1	0	1	0	Self clocked, external 32x	0-187.5K
1	1	0	0	Self clocked, internal fixed	375K
1	1	1	0	Self clocked, internal fixed	187.5K

\*Based on a 12 Mhz crystal frequency

\*\*0-1M bps in loop configuration

### STS: Status/Command Register (bit-addressable)

Bit: 7	6	5	4	3	2	1	0
TBF	RBE	RTS	SI	BOV	OPB	AM	RBP

The Status/Command Register (Address C8H) provides operational control of the SIU by the 8044 CPU, and enables the SIU to post status information for the CPU's access. The SIU can read STS, and can alter certain bits, as indicated below. The CPU can both read and write STS asynchronously. However, 2-cycle instructions that access STS during both cycles ('JBC/B, REL' and 'MOV /B.C.') should not be used, since the SIU may write to STS between the two CPU accesses.

The individual bits of the Status/Command Register are as follows:

Bit #	Name	Description
STS.0	RBP	Receive Buffer Protect. Inhibits writing of data into the receive buffer. In AUTO mode, RBP forces an RNR response instead of an RR.
STS.1	AM	AUTO Mode/Addressed Mode. Selects AUTO mode where AUTO mode is allowed. If NB is true, (=1), the AM bit selects the addressed mode. AM may be cleared by the SIU.
STS.2	OPB	Optional Poll Bit. Determines whether the SIU will generate an AUTO response to an optional poll (UP with P=0). OPB may be set or cleared by the SIU.
STS.3	BOV	Receive Buffer Overrun. BOV may be set or cleared by the SIU.
STS.4	SI	SIU Interrupt. This is one of the five interrupt sources to the CPU. The vector location = 23H. SI may be set by the SIU. It should be cleared by the CPU before returning from an interrupt routine.
STS.5	RTS	Request To Send. Indicates that the 8044 is ready to transmit or is transmitting. RTS may be read or written by the CPU. RTS may be read by the SIU, and in AUTO mode may be written by the SIU.



**STS.6 RBE** Receive Buffer Empty. RBE can be thought of as Receive Enable. RBE is set to one by the CPU when it is ready to receive a frame, or has just read the buffer, and to zero by the SIU when a frame has been received.

**STS.7 TBF** Transmit Buffer Full. Written by the CPU to indicate that it has filled the transmit buffer. TBF may be cleared by the SIU.

#### NSNR: Send/Receive Count Register (bit-addressable)

Bit: 7 6 5 4 3 2 1 0  
 NS2 NS1 NS0 SES NR2 NR1 NR0 SER

The Send/Receive Count Register (Address D8H) contains the transmit and receive sequence numbers, plus tally error indications. The SIU can both read and write NSNR. The 8044 CPU can both read and write NSNR asynchronously. However, 2-cycle instructions that access NSNR during both cycles ('JBC /B, REL', and 'MOV /B,C') should not be used, since the SIU may write to NSNR between the two 8044 CPU accesses.

The individual bits of the Send/Receive Count Register are as follows:

Bit #	Name	Description
NSNR.0	SER	Receive Sequence Error: NS (P) $\neq$ NR (S)
NSNR.1	NR0	Receive Sequence Counter—Bit 0
NSNR.2	NR1	Receive Sequence Counter—Bit 1
NSNR.3	NR2	Receive Sequence Counter—Bit 2
NSNR.4	SES	Send Sequence Error: NR (P) $\neq$ NS (S) and NR (P) $\neq$ NS (S) + 1
NSNR.5	NS0	Send Sequence Counter — Bit 0
NSNR.6	NS1	Send Sequence Counter — Bit 1
NSNR.7	NS2	Send Sequence Counter — Bit 2

#### Parameter Registers

There are eight parameter registers that are used in connection with SIU operation. All eight registers may be read or written by the 8044 CPU. RFL and RCB are normally loaded by the SIU.

The eight parameter registers are as follows:

#### STAD: Station Address Register (byte-addressable)

The Station Address register (Address CEH) contains the station address. To prevent access conflict, the CPU should access STAD only when the SIU is idle (RTS=0

and RBE=0). Normally, STAD is accessed only during initialization.

#### TBS: Transmit Buffer Start Address Register (byte-addressable)

The Transmit Buffer Start address register (Address DCH) points to the location in on-chip RAM for the beginning of the I-field of the frame to be transmitted. The CPU should access TBS only when the SIU is not transmitting a frame (when TBF=0).

#### TBL: Transmit Buffer Length Register (byte-addressable)

The Transmit Buffer Length register (Address DBH) contains the length (in bytes) of the I-field to be transmitted. A blank I-field (TBL=0) is valid. The CPU should access TBL only when the SIU is not transmitting a frame (when TBF=0).

**NOTE:** The transmit and receive buffers are not allowed to "wrap around" in the on-chip RAM. A "buffer end" is automatically generated if address 191 (BFH) is reached.

#### TCB: Transmit Control Byte Register (byte-addressable)

The Transmit Control Byte register (Address DAH) contains the byte which is to be placed in the control field of the transmitted frame, during NON-AUTO mode transmission. The CPU should access TCB only when the SIU is not transmitting a frame (when TBF=0). The  $N_S$  and  $N_R$  counters are not used in the NON-AUTO mode.

#### RBS: Receive Buffer Start Address Register (byte-addressable)

The Receive Buffer Start address register (Address CCH) points to the location in on-chip RAM where the beginning of the I-field of the frame being received is to be stored. The CPU should write RBS only when the SIU is not receiving a frame (when RBE=0).

#### RBL: Receive Buffer Length Register (byte-addressable)

The Receive Buffer Length register (Address CBH) contains the length (in bytes) of the area in on-chip

RAM allocated for the received I-field. RBL=0 is valid. The CPU should write RBL only when RBE=0.

#### RFL: Receive Field Length Register (byte-addressable)

The Received Field Length register (Address CDH) contains the length (in bytes) of the received I-field that has just been loaded into on-chip RAM. RFL is loaded by the SIU. RFL=0 is valid. RFL should be accessed by the CPU only when RBE=0.

**RCB: Receive Control Byte Register**  
(byte-addressable)

The Received Control Byte register (Address CAH) contains the control field of the frame that has just been received. RCB is loaded by the SIU. The CPU can only read RCB, and should only access RCB when RBE=0.

## ICE Support Registers

The 8044 In-Circuit Emulator (ICE-44) allows the user to exercise the 8044 application system and monitor the execution of instructions in real time.

The emulator operates with Intel's Intellec® development system. The development system interfaces with the user's 8044 system through an in-cable buffer box. The cable terminates in a 8044 pin-compatible plug, which fits into the 8044 socket in the user's system. With

the emulator plug in place, the user can exercise his system in real time while collecting up to 255 instruction cycles of real-time data. In addition, he can single-step the program.

Static RAM is available (in the in-cable buffer box) to emulate the 8044 internal and external program memory and external data memory. The designer can display and alter the contents of the replacement memory in the buffer box, the internal data memory, and the internal 8044 registers, including the SFRs.

**SIUST: SIU State Counter (byte-addressable)**

The SIU State Counter (Address D9H) reflects the state of the internal logic which is under SIU control. Therefore, care must be taken not to write into this register. This register provides a useful means for debugging 8044 receiver problem.

## ABSOLUTE MAXIMUM RATINGS\*

Ambient Temperature Under Bias..... 0 to 70°C  
 Storage Temperature..... - 65°C to + 150°C  
 Voltage on Any Pin With  
 Respect to Ground (V<sub>SS</sub>)..... - 0.5V to + 7V  
 Power Dissipation..... 2 Watts

\*Notice: Stresses above those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.

DC CHARACTERISTICS (T<sub>A</sub> = 0°C to 70°C, V<sub>CC</sub> = 5V ± 10%, V<sub>SS</sub> = 0V)

Symbol	Parameter	Min	Typ	Max	Units	Test Conditions
V <sub>IL</sub>	Input Low Voltage	- 0.5		0.8	V	
V <sub>IH</sub>	Input High Voltage (Except RST/VPD and XTAL2)	2.0		V <sub>CC</sub> + 0.5	V	
V <sub>IH1</sub>	Input High Voltage To RST/VPD For Reset, XTAL2	2.5			V	XTAL1 to V <sub>SS</sub>
V <sub>OL</sub>	Output Low Voltage Ports 1, 2, 3 (Note 1)			0.45	V	IOL = 1.6mA
V <sub>OL1</sub>	Output Low Voltage Port 0 ALE, $\overline{\text{PSEN}}$ (Note 1)			0.45	V	IOL = 3.2mA
V <sub>OH</sub>	Output High Voltage Ports 1, 2, 3	2.4			V	IOH = -80μA
V <sub>OH1</sub>	Output High Voltage Port 0, ALE, $\overline{\text{PSEN}}$	2.4			V	IOH = -400μA
I <sub>IL</sub>	Logical 0 Input Current Ports 1, 2, 3			- 800	μA	XTAL1 at V <sub>SS</sub> V <sub>IL</sub> = 0.45V
I <sub>IH1</sub>	Input High Current To RST/VPD For Reset			500	μA	V <sub>in</sub> = V <sub>CC</sub> - 1.5V
I <sub>LI</sub>	Input Leakage Current To Port 0, $\overline{\text{EA}}$			10	μA	0.45V < V <sub>in</sub> < V <sub>CC</sub>
I <sub>CC</sub>	Power Supply Current		125	200	mA	T <sub>A</sub> = 25°C
C <sub>IO</sub>	Capacitance of I/O Buffer			10	pF	f <sub>c</sub> = 1MHz
I <sub>IL2</sub>	Logical 0 Input Current XTAL 2			-3.5	mA	XTAL1 = V <sub>SS</sub> V <sub>IL</sub> = 0.45V

Note 1: V<sub>OL</sub> is degraded when the RUP1-44 rapidly discharges external capacitance. This A.C. noise is most pronounced during emission of address data. When using external memory, locate the latch or buffer as close to the RUP1-44 as possible.

Datum	Emitting Ports	Degraded I/O Lines	VOL (peak) (max)
Address	P2, P0	P1, P3	.8V
Write Data	P0	P1, P3, ALE	.8V

**A.C. CHARACTERISTICS** (TA 0°C to 70°C, VCC = 5V ± 10%, VSS = 0V, CL for Port 0, ALE and PSEN Outputs = 100pF;  
CL for All Other Outputs = 80 pF)

**Program Memory**

Symbol	Parameter	12 MHz Clock			Variable Clock 1/TCLCL = 1.2 MHz to 12 MHz		
		Min	Max	Units	Min	Max	Units
TLHLL	ALE Pulse Width	127		ns	2TCLCL-40		ns
TAVLL	Address Setup to ALE	43		ns	TCLCL-40		ns
TLLAX <sup>1</sup>	Address Hold After ALE	48		ns	TCLCL-35		ns
TLLIV	ALE To Valid Instr In		233	ns		4TCLCL-100	ns
TLLPL	ALE To PSEN	58		ns	TCLCL-25		ns
TPLPH	PSEN Pulse Width	215		ns	3TCLCL-35		ns
TPLIV	PSEN To Valid Instr In		125	ns		3TCLCL-125	ns
TPXIX	Input Instr Hold After PSEN	0		ns	0		ns
TPXIZ <sup>2</sup>	Input Instr Float After PSEN		63	ns		TCLCL-20	ns
TPXAV <sup>2</sup>	Address Valid After PSEN	75		ns	TCLCL-8		ns
TAVIV	Address To Valid Instr In		302	ns		5TCLCL-115	ns
TAZPL	Address Float To PSEN	-25		ns	-25		ns

**Notes:**

1. TLLAX for access to program memory is different from TLLAX for data memory.
2. Interfacing RUP1-44 devices with float times up to 75ns is permissible. This limited bus contention will not cause any damage to Port 0 drivers.

**External Data Memory**

Symbol	Parameter	12 MHz Clock			Variable Clock 1/TCLCL = 1.2 MHz to 12 MHz		
		Min	Max	Units	Min	Max	Units
TRLRH	RD Pulse Width	400		ns	6TCLCL-100		ns
TWLWH	WR Pulse Width	400		ns	6TCLCL-100		ns
TLLAX <sup>1</sup>	Address Hold After ALE	48		ns	TCLCL-35		ns
TRLDV	RD To Valid Data In		250	ns		5TCLCL-165	ns
TRHDX	Data Hold After RD	0		ns	0		ns
TRHDZ	Data Float After RD		97	ns		2TCLCL-70	ns
TLLDV	ALE To Valid Data In		517	ns		8TCLCL-150	ns
TAVDV	Address To Valid Data In		585	ns		9TCLCL-165	ns
TLLWL	ALE To WR or RD	200	300	ns	3TCLCL-50	3TCLCL + 50	ns
TAVWL	Address To WR or RD	203		ns	4TCLCL-130		ns
TWHLH	WR or RD High To ALE High	43	123	ns	TCLCL-40	TCLCL + 40	ns
TDVWX	Data Valid To WR Transition	33		ns	TCLCL-50		ns
TQVWH	Data Setup Before WR	433		ns	7TCLCL-150		ns
TWHQX	Data Hold After WR	33		ns	TCLCL-50		ns
TRLAZ	Address Float After RD		0	ns		0	ns

**Note 1.** TLLAX for access to program memory is different from TLLAX for access data memory.

**Serial Interface**

Symbol	Parameter	Min	Max	Units
TDCL	Data Clock	420		ns
TDCL	Data Clock Low	180		ns
TDCH	Data Clock High	100		ns

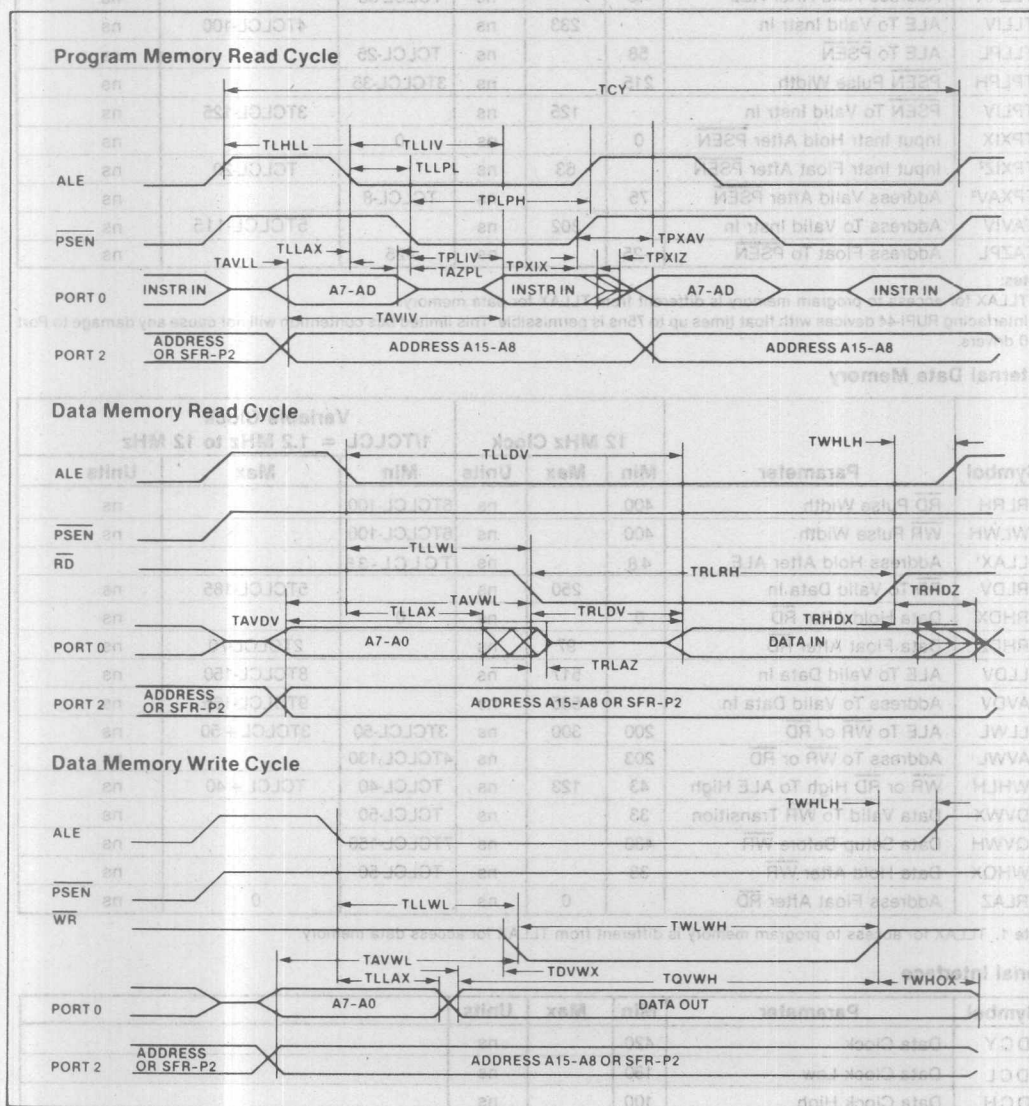


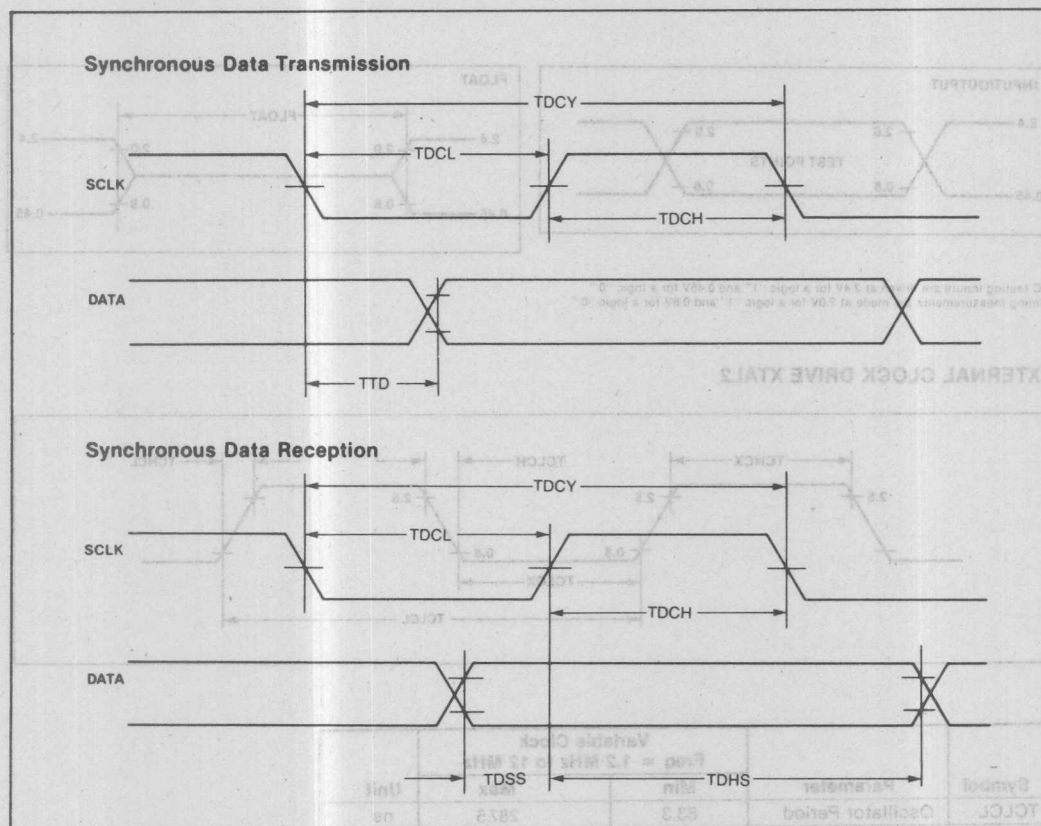
### Serial Interface (Continued)

tTD	Transmit Data Delay		180	ns
tDSS	Data Setup Time	40		ns
tDHS	Data Hold Time	40		ns

## WAVEFORMS

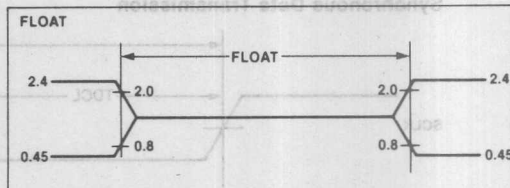
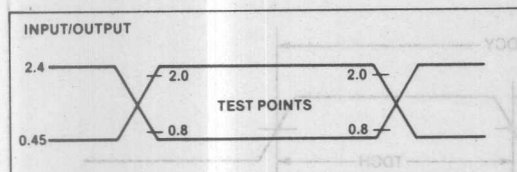
## Memory Access





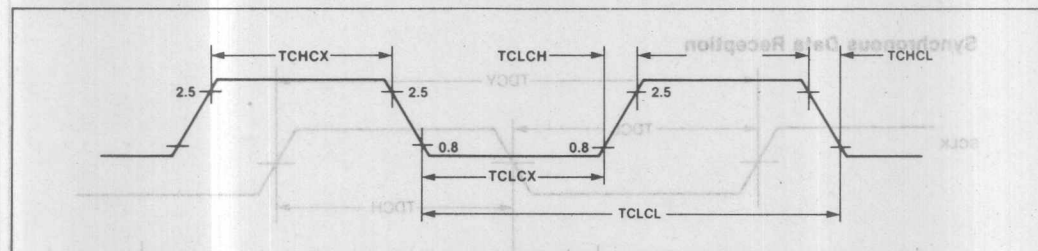
Symbol	Parameter	Unit
TDCY	Oscillator Period	ns
TDCL	High Time	ns
TDCH	Low Time	ns
TDSS	Rise Time	ns
TDHS	Fall Time	ns

## AC TESTING INPUT, OUTPUT, FLOAT WAVEFORMS



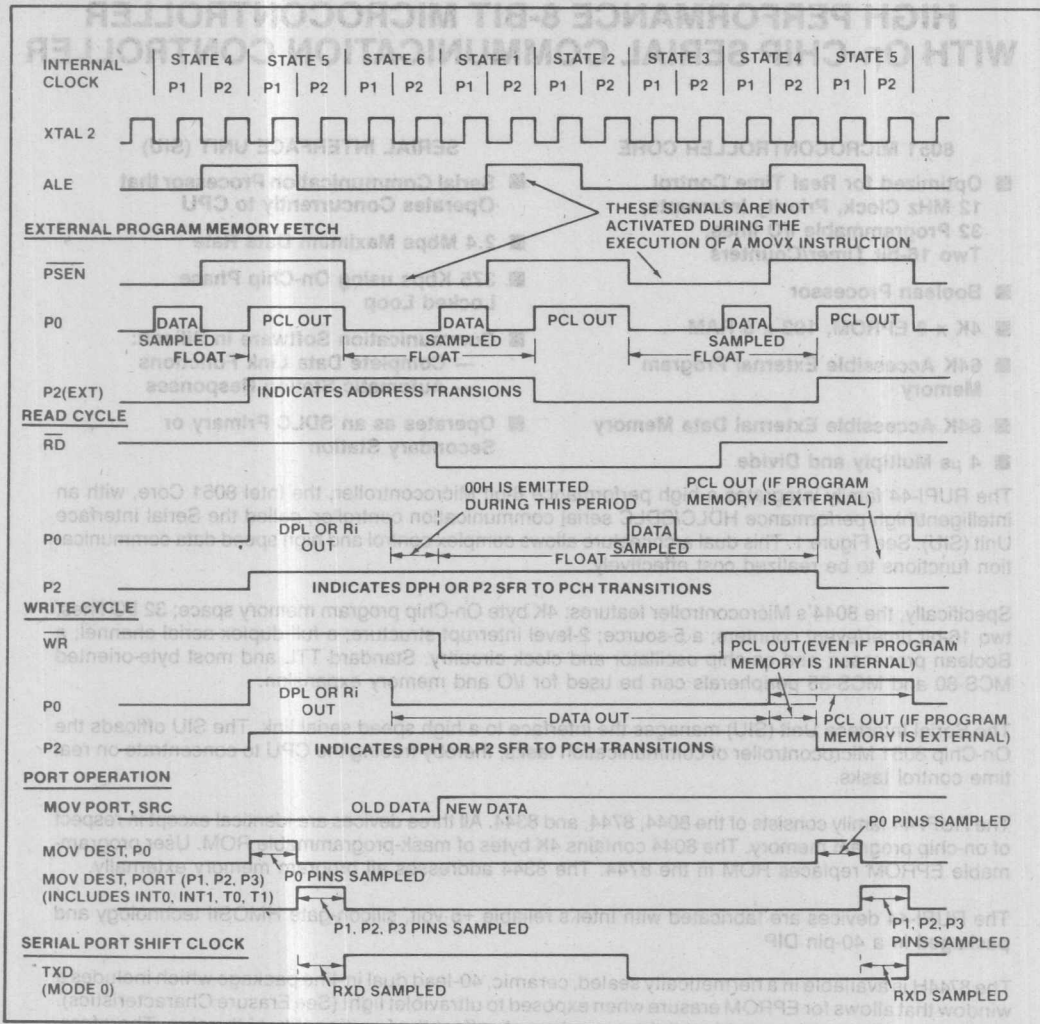
AC testing inputs are driven at 2.4V for a logic "1" and 0.45V for a logic "0".  
Timing measurements are made at 2.0V for a logic "1" and 0.8V for a logic "0".

## EXTERNAL CLOCK DRIVE XTAL2



Symbol	Parameter	Variable Clock Freq = 1.2 MHz to 12 MHz		Unit
		Min	Max	
TCLCL	Oscillator Period	83.3	287.5	ns
TCHCX	High Time	20	TCLCL-TCLCX	ns
TCLCX	Low Time	20	TCLCL-TCHCX	ns
TCLCH	Rise Time		20	ns
TCHCL	Fall Time		20	ns

# CLOCK WAVEFORMS



This diagram indicates when signals are clocked internally. The time it takes the signals to propagate to the pins, however, ranges from 25 to 125 ns. This propagation delay is dependent on variables such as temperature and pin loading. Propagation also varies from output to output and component to component. Typically though, ( $T_A = 25^\circ\text{C}$ , fully loaded) RD and WR propagation delays are approximately 50 ns. The other signals are typically 85 ns. Propagation delays are incorporated in the AC specifications.



Figure 1. Dual Controller Architecture



## 8744H

# HIGH PERFORMANCE 8-BIT MICROCONTROLLER WITH On-CHIP SERIAL COMMUNICATION CONTROLLER

### 8051 MICROCONTROLLER CORE

- Optimized for Real Time Control  
12 MHz Clock, Priority Interrupts,  
32 Programmable I/O lines,  
Two 16-bit Timer/Counters
- Boolean Processor
- 4K x 8 EPROM, 192 x 8 RAM
- 64K Accessible External Program Memory
- 64K Accessible External Data Memory
- 4  $\mu$ s Multiply and Divide

### SERIAL INTERFACE UNIT (SIU)

- Serial Communication Processor that Operates Concurrently to CPU
- 2.4 Mbps Maximum Data Rate
- 375 Kbps using On-Chip Phase Locked Loop
- Communication Software in Silicon:
  - Complete Data Link Functions
  - Automatic Station Responses
- Operates as an SDLC Primary or Secondary Station

The RUP1-44 family integrates a high performance 8-bit Microcontroller, the Intel 8051 Core, with an intelligent/high performance HDLC/SDLC serial communication controller, called the Serial Interface Unit (SIU). See Figure 1. This dual architecture allows complex control and high speed data communication functions to be realized cost effectively.

Specifically, the 8044's Microcontroller features: 4K byte On-Chip program memory space; 32 I/O lines; two 16-bit timer/event counters; a 5-source; 2-level interrupt structure; a full duplex serial channel; a Boolean processor; and on-chip oscillator and clock circuitry. Standard TTL and most byte-oriented MCS-80 and MCS-85 peripherals can be used for I/O and memory expansion.

The Serial Interface Unit (SIU) manages the interface to a high speed serial link. The SIU offloads the On-Chip 8051 Microcontroller of communication tasks, thereby freeing the CPU to concentrate on real time control tasks.

The RUP1-44 family consists of the 8044, 8744, and 8344. All three devices are identical except in respect of on-chip program memory. The 8044 contains 4K bytes of mask-programmable ROM. User programmable EPROM replaces ROM in the 8744. The 8344 addresses all program memory externally.

The RUP1-44 devices are fabricated with Intel's reliable +5 volt, silicon-gate HMOSII technology and packaged in a 40-pin DIP

The 8744H is available in a hermetically sealed, ceramic, 40-lead dual in-line package which includes a window that allows for EPROM erasure when exposed to ultraviolet light (See Erasure Characteristics). During normal operation, ambient light may adversely affect the functionality of the chip. Therefore, applications which expose the 8744H to ambient light may require an opaque label over the window.

### 8044's Dual Controller Architecture

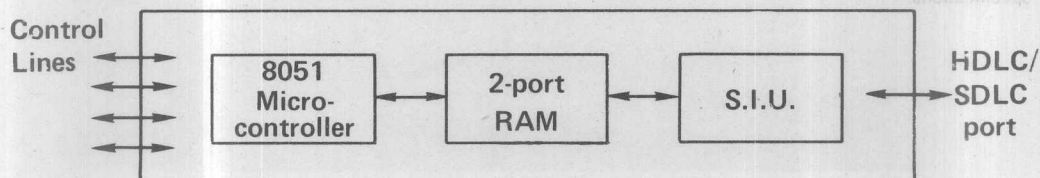


Figure 1. Dual Controller Architecture

Table 1. RUP1™ -44 Family Pin Description

**VSS**

Circuit ground potential.

**VCC**

+5V power supply during operation and program verification.

**PORT 0**

Port 0 is an 8-bit open drain bidirectional I/O port. It is also the multiplexed low-order address and data bus when using external memory. It is used for data output during program verification. Port 0 can sink/source eight LS TTL loads.

**PORT 1**

Port 1 is an 8-bit quasi-bidirectional I/O port. It is used for the low-order address byte during program verification. Port 1 can sink/source four LS TTL loads.

In non-loop mode two of the I/O lines serve alternate functions:

- RTS (P1.6). Request-to-Send output. A low indicates that the RUP1-44 is ready to transmit.
- CTS (P1.7). Clear-to-Send input. A low indicates that a receiving station is ready to receive.

**PORT 2**

Port 2 is an 8-bit quasi-bidirectional I/O port. It also emits the high-order address byte when accessing external memory. It is used for the high-order address and the control signals during program verification. Port 2 can sink/source four LS TTL loads.

**PORT 3**

Port 3 is an 8-bit quasi-bidirectional I/O port. It also contains the interrupt, timer, serial port and RD and WR pins that are used by various options. The output latch corresponding to a secondary function must be programmed to a one (1) for that function to operate. Port 3 can sink/source LS TTL loads.

In addition to I/O, some of the pins also serve alternate functions as follows:

- I/O RxD (P3.0). In point-to-point or multipoint configurations, this pin controls the direction of pin P3.1. Serves as Receive Data input in loop and diagnostic modes.
- DATA TxD (P3.1). In point-to-point or multipoint configurations, this pin functions as data input/output. In loop mode, it serves as transmit pin. A '0' written to this pin enables diagnostic mode.
- INT0 (P3.2). Interrupt 0 input or gate control input for counter 0.
- INT1 (P3.3). Interrupt 1 input or gate control input for counter 1.
- TO (P3.4). Input to counter 0.

- SCLK T1 (P3.5). In addition to I/O, this pin provides input to counter 1 or serves as SCLK (serial clock) input.
- WR (P3.6). The write control signal latches the data byte from Port 0 into the External Data Memory.
- RD (P3.7). The read control signal enables External Data Memory to Port 0.

**RST**

A high on this pin for two machine cycles while the oscillator is running resets the device. A small external pulldown resistor ( $\approx 8.2k\Omega$ ) from RST to VSS permits power-on reset when a capacitor ( $\approx 10\mu f$ ) is also connected from this pin to Vcc.

**ALE/PROG**

Provides Address Latch Enable output used for latching the address into external memory during normal operation. It is activated every six oscillator periods except during an external data memory access. It also receives the program pulse input for programming the EPROM version.

**PSEN**

The Program Store Enable output is a control signal that enables the external Program Memory to the bus during external fetch operations. It is activated every six oscillator periods, except during external data memory accesses. Remains high during internal program execution.

**EA/VPP**

When held at a TTL high level, the RUP1-44 executes instructions from the internal ROM when the PC is less than 4096. When held at a TTL low level, the RUP1-44 fetches all instructions from external Program Memory. The pin also receives the 21V EPROM programming supply voltage on the 8744.

**XTAL 1**

Input to the oscillator's high gain amplifier. Required when a crystal is used. Connect to VSS when external source is used on XTAL 2.

**XTAL 2**

Output from the oscillator's amplifier. Input to the internal timing circuitry. A crystal or external source can be used.

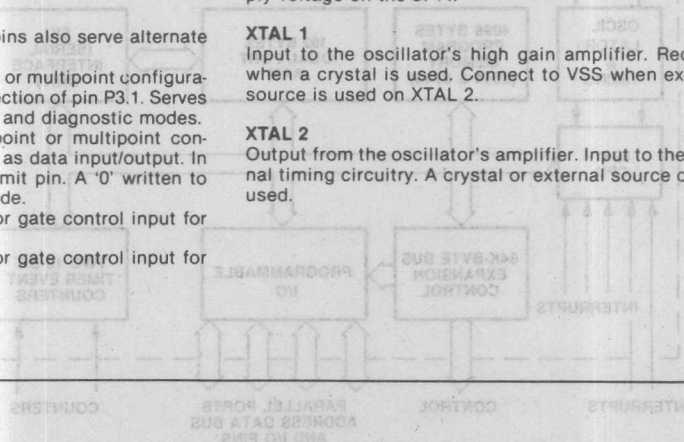


Figure 4  
Block Diagram

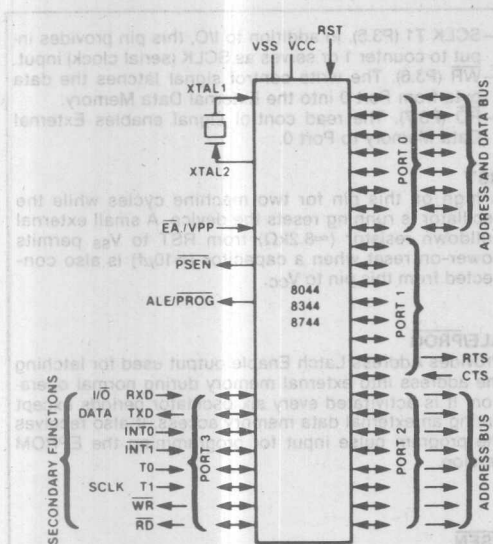


Figure 2.  
Logic Symbol

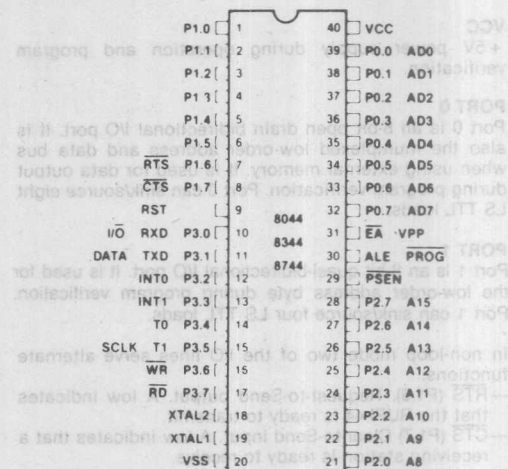


Figure 3. Pin  
Configuration

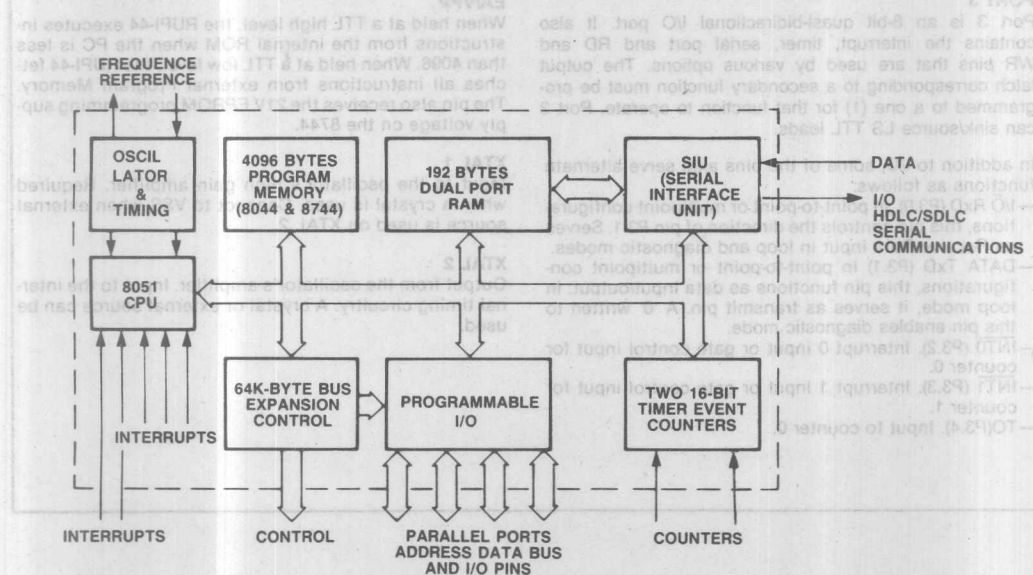


Figure 4.  
Block Diagram

**ABSOLUTE MAXIMUM RATINGS\***

Ambient Temperature Under Bias . . . 0°C to 70°C

Storage Temperature . . . . . -65°C to +150°C

Voltage On Any Pin to V<sub>SS</sub>(Except V<sub>PP</sub>) . . . . . -0.5V to +7VVoltage from V<sub>PP</sub> to V<sub>SS</sub> . . . . . 21.5V

Power Dissipation . . . . . 2 W

\*NOTICE: Stresses above those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.

**D.C. CHARACTERISTICS:** (T<sub>A</sub> = 0°C to 70°C; V<sub>CC</sub> = 4.5V to 5.5V, V<sub>SS</sub> = 0V)

Symbol	Parameter	Min	Max	Unit	Test Conditions
V <sub>IL</sub>	Input Low Voltage (except $\overline{EA}$ )	-0.5	0.8	V	
V <sub>IL1</sub>	Input Low Voltage to $\overline{EA}$	0	0.8	V	
V <sub>IH</sub>	Input High Voltage (Except XTAL2, RST)	2.0	V <sub>CC</sub> + 0.5	V	
V <sub>IH1</sub>	Input High Voltage to XTAL2, RST	2.5	V <sub>CC</sub> + 0.5	V	XTAL1 = V <sub>SS</sub>
V <sub>OL</sub>	Output Low Voltage Ports 1, 2, 3 (Note 1)		0.45	V	I <sub>OL</sub> = 1.6mA
V <sub>OL1</sub>	Output Low Voltage Port 0, ALE, $\overline{PSEN}$ (Note 1)		0.60 0.45	V	I <sub>OL</sub> = 3.2mA I <sub>OL</sub> = 2.4mA
V <sub>OH</sub>	Output High Voltage Ports 1, 2, 3	2.4		V	I <sub>OH</sub> = -80μA
V <sub>OH1</sub>	Output High Voltage Port 0 (in External Bus Mode), ALE, $\overline{PSEN}$	2.4		V	I <sub>OH</sub> = -400μA
I <sub>IL</sub>	Logical 0 Input Current P1, P2, P3		500	μA	V <sub>in</sub> = 0.45V
I <sub>IL1</sub>	Logical 0 Input Current to $\overline{EA}/V_{PP}$		-15	mA	V <sub>in</sub> = 0.45V
I <sub>IL2</sub>	Logical 0 Input Current to XTAL2		-3.5	mA	XTAL1 = V <sub>SS</sub> V <sub>in</sub> = 0.45V
I <sub>LI</sub>	Input Leakage Current to Port 0		100	μA	0.45 < V <sub>in</sub> < V <sub>CC</sub>
I <sub>IH</sub>	Logical Input Current to $\overline{EA}/V_{PP}$		500	μA	V <sub>in</sub> = 2.4V
I <sub>IH1</sub>	Input Current to RST/V <sub>PD</sub> to activate reset		500	μA	V <sub>in</sub> < (V <sub>CC</sub> - 1.5V)
I <sub>CC</sub>	Power Supply Current		270	mA	All outputs disconnected, EA = V <sub>CC</sub>
C <sub>IO</sub>	Capacitance of I/O Buffers		10	pF	f <sub>c</sub> = 1 MHz T <sub>A</sub> = 25°C

ns			48		Address Hold After ALE	TLJAX
ns	25	325			RD to Valid Data In	TRLDV
ns	0	0			Data Hold After RD	TRHDZ
ns	70	97			Data Hold After RD	TRHDZ
ns	150	217			ALE to Valid Data In	TLJDV
ns	185	265			Address to Valid Data In	TAVDV
ns	150	300	500		ALE to WR or RD	TLJWJ
ns	130	303			Address to WR or RD	TAVWJ
ns	70	13			Data Valid to WR Transition	TDVWJ
ns	150	433			Data Setup to WR High	TDVWJ
ns	20	33			Data Hold After WR	TDVWJ
ns	25	25			RD Low to Address Float	TRJAZ
ns	50	133	33		RD or WR High to ALE High	TDVWJ



**AC CHARACTERISTICS** ( $T_A = 0^\circ\text{C}$  to  $70^\circ\text{C}$ ,  $V_{CC} = 4.5\text{V}$  to  $5.5\text{V}$ ,  $V_{SS} = 0\text{V}$ , Load Capacitance for Port 0, ALE, and  $\overline{\text{PSEN}} = 100\text{ pF}$ ; Load Capacitance for all other outputs =  $80\text{ pF}$ )

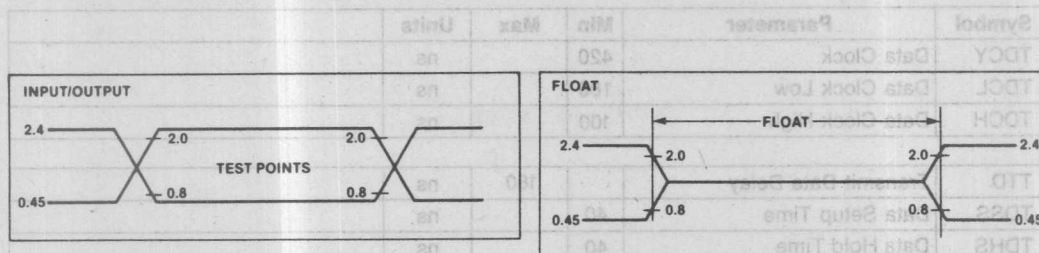
# **EXTERNAL PROGRAM MEMORY CHARACTERISTICS**

Symbol	Parameter	8744H 12 MHz Osc		Variable Oscillator		Units
		Min	Max	Min	Max	
TCLCL	Oscillator Period			83.3	285.7	ns
TLHLL	ALE Pulse Width	127		2TCLCL-40		ns
TAVLL	Address Valid to ALE	53		TCLCL-40		ns
TLLAX	Address Hold after ALE	48		TCLCL-35		ns
TLLIV	ALE to Valid Instr In	0	183	4TCLCL-150		ns
TLLPL	ALE to $\overline{\text{PSEN}}$	58		TCLCL-25		ns
TPLPH	$\overline{\text{PSEN}}$ Pulse Width	190		3TCLCL-60		ns
TPLIV	$\overline{\text{PSEN}}$ to Valid Instr In		100	3TCLCL-150		ns
TPXIX	Input Instr Hold after $\overline{\text{PSEN}}$	0		0		ns
TPXIZ	Input Instr Float after $\overline{\text{PSEN}}$		63	TCLCL-20		ns
TPXAV	$\overline{\text{PSEN}}$ to Address Valid	75		TCLCL-8		ns
TAVIV	Address to Valid Instr In		267	5TCLCL-150		ns
TAZPL	Address Float to $\overline{\text{PSEN}}$	-25		-25		ns

# **EXTERNAL DATA MEMORY CHARACTERISTICS**

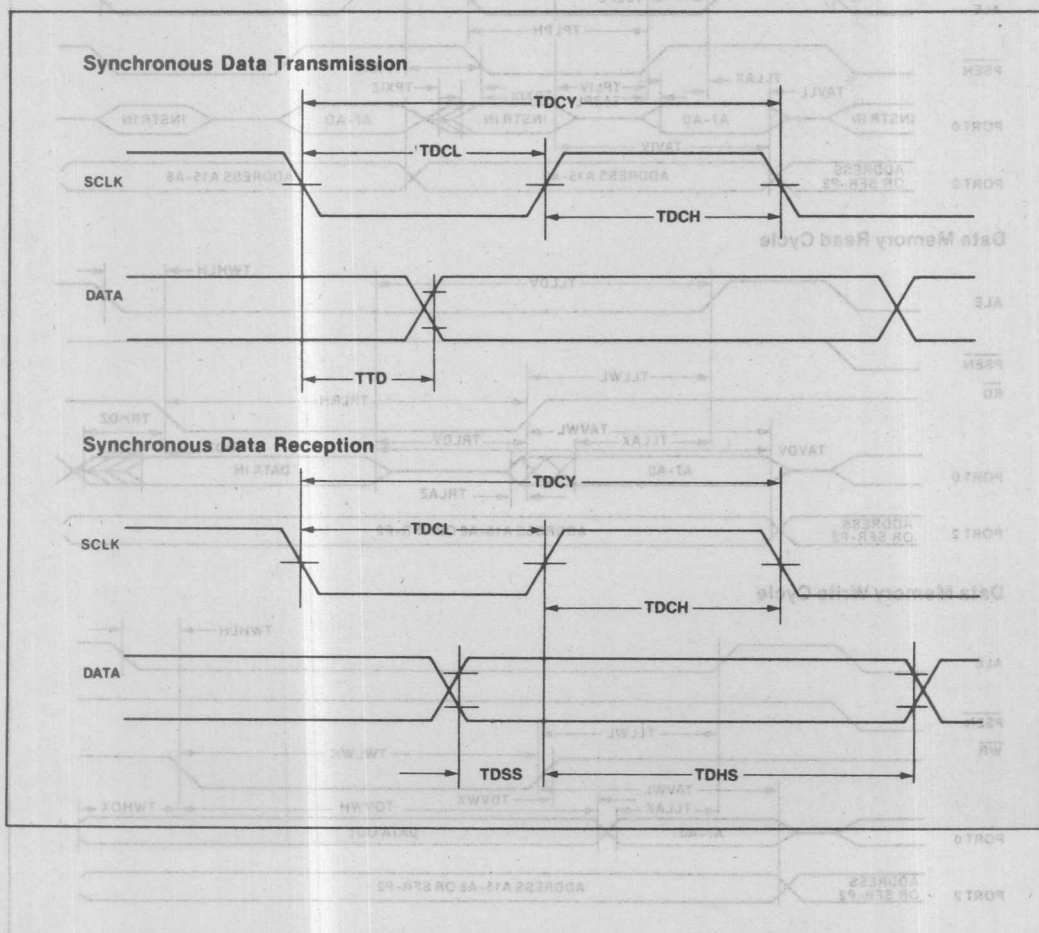
Symbol	Parameter	8744H 12 MHz Osc		Variable Oscillator		Units
		Min	Max	Min	Max	
TRLRH	$\overline{\text{RD}}$ Pulse Width	400		6TCLCL-100		ns
TWLWH	$\overline{\text{WR}}$ Pulse Width	400		6TCLCL-100		ns
TLLAX	Address Hold After ALE	48		TCLCL-35		ns
TRLDV	$\overline{\text{RD}}$ to Valid Data In		252		5TCLCL-165	ns
TRHDX	Data Hold after $\overline{\text{RD}}$	0		0		ns
TRHDZ	Data Float after $\overline{\text{RD}}$		97		2TCLCL-70	ns
TLLDV	ALE to Valid Data In		517		8TCLCL-150	ns
TAVDV	Address to Valid Data In		585		9TCLCL-165	ns
TLLWL	ALE to $\overline{\text{WR}}$ or $\overline{\text{RD}}$	200	300	3TCLCL-50	3TCLCL+50	ns
TAVWL	Address to $\overline{\text{WR}}$ or $\overline{\text{RD}}$	203		4TCLCL-130		ns
TQVWX	Data Valid to $\overline{\text{WR}}$ Transition	13		TCLCL-70		ns
TQVWH	Data Setup to $\overline{\text{WR}}$ High	433		7TCLCL-150		ns
TWHQX	Data Held after $\overline{\text{WR}}$	33		TCLCL-50		ns
TRLAZ	$\overline{\text{RD}}$ Low to Address Float		25		25	ns
TWHLH	$\overline{\text{RD}}$ or $\overline{\text{WR}}$ High to ALE High	33	133	TCLCL-50	TCLCL+50	ns

# AC TESTING INPUT, OUTPUT, FLOAT WAVEFORMS



AC testing inputs are driven at 2.4V for a logic 1 and 0.45V for a logic 0  
 Timing measurements are made at 2.0V for a logic 1 and 0.8V for a logic 0  
 For timing purposes, the float state is defined as the point at which a P0 pin sinks 3.2mA or sources 400 $\mu$ A at the voltage test levels

## SERIAL I/O WAVEFORMS

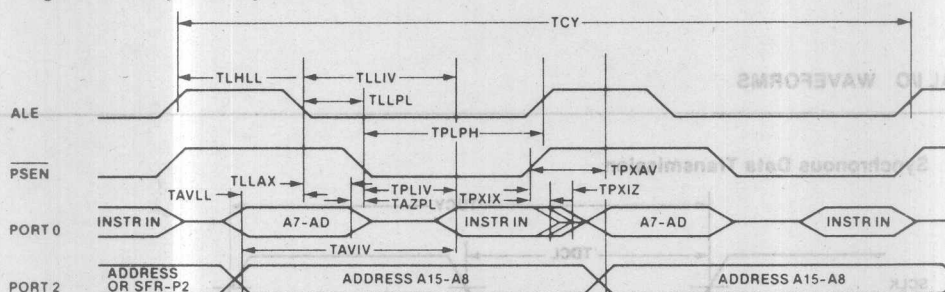


## Serial Interface

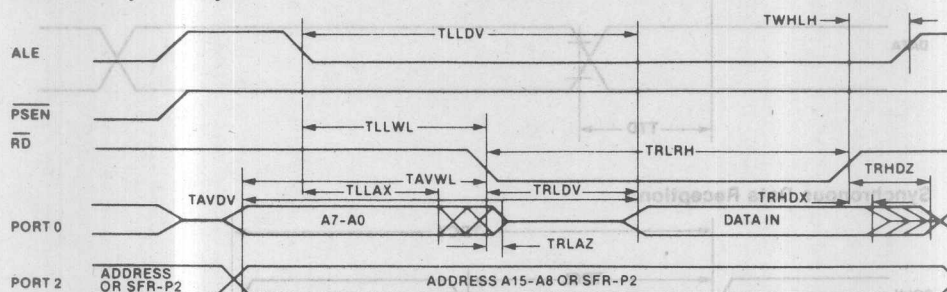
Symbol	Parameter	Min	Max	Units	
TDCY	Data Clock	420		ns	
TDCL	Data Clock Low	180		ns	
TDCH	Data Clock High	100		ns	
TTD	Transmit Data Delay		180	ns	
TDSS	Data Setup Time	40		ns	
TDHS	Data Hold Time	40		ns	

## Memory Access

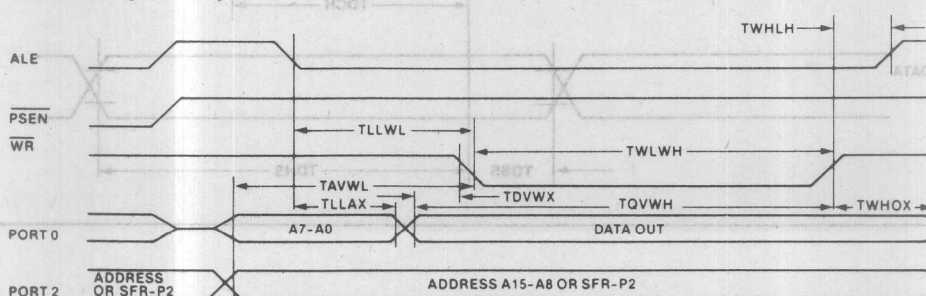
### Program Memory Read Cycle



### Data Memory Read Cycle



### Data Memory Write Cycle

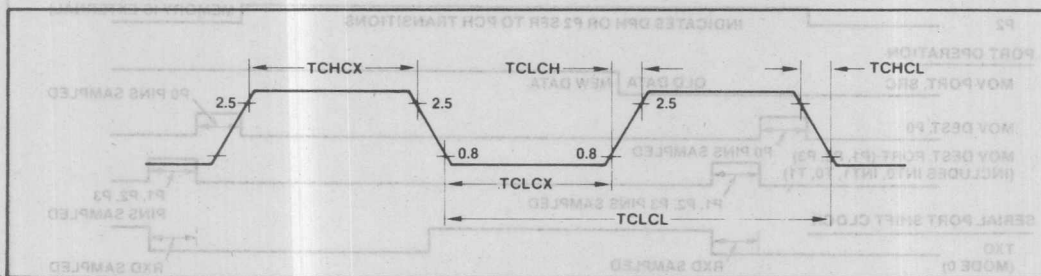


**NOTE:** VOL is degraded when the 8744H rapidly discharges external capacitance. This AC noise is most pronounced during emission of address data. When using external memory, locate the latch or buffer as close to the 8744H as possible.

Datum	Emitting Ports	Degraded I/O Lines	VOL (peak) (max)
Address	P2, P0	P1, P3	0.8V
Write Data	P0	P1, P3, ALE	0.8V

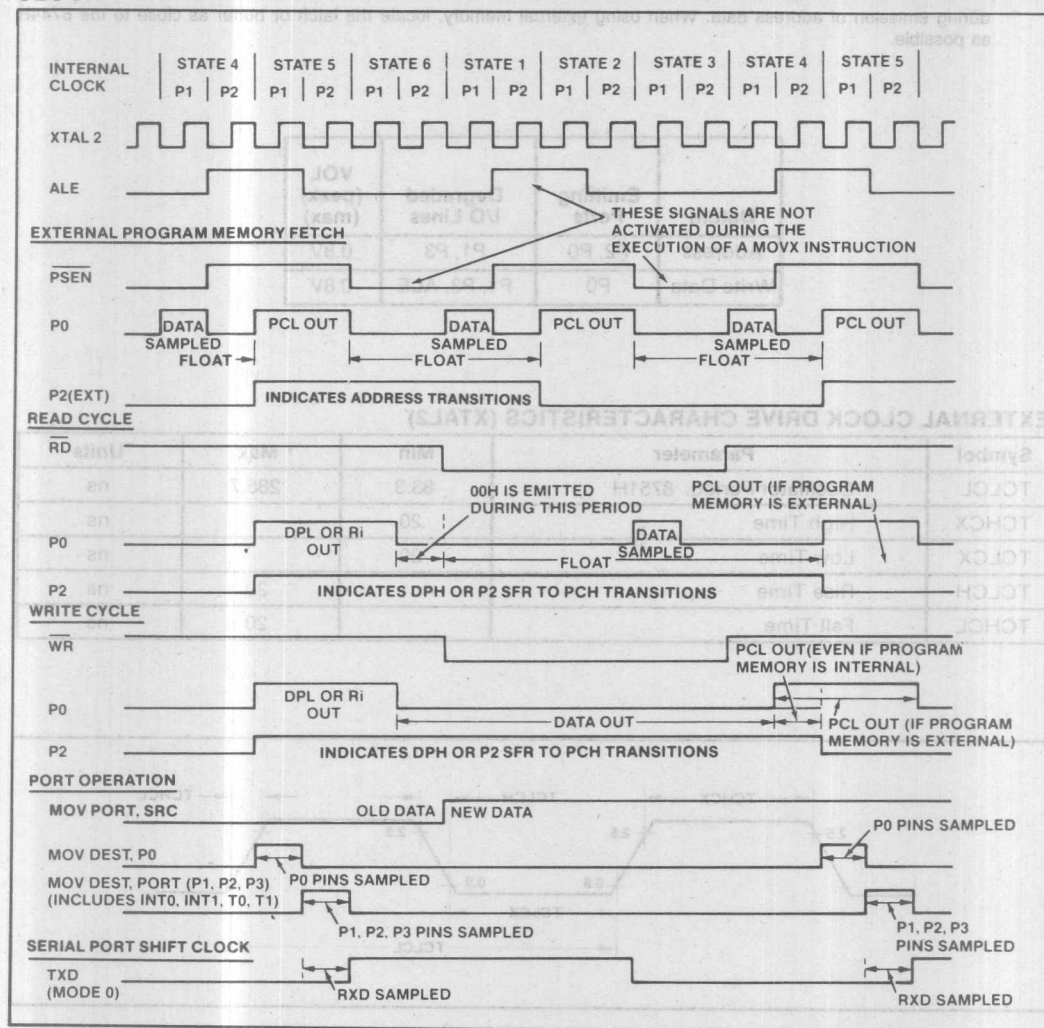
### EXTERNAL CLOCK DRIVE CHARACTERISTICS (XTAL2)

Symbol	Parameter	Min	Max	Units
TCLCL	Oscillator Period: 8751H	83.3	285.7	ns
TCHCX	High Time	20		ns
TCLCX	Low Time	20		ns
TCLCH	Rise Time		20	ns
TCHCL	Fall Time		20	ns



This diagram indicates when signals are clocked internally. The time it takes the signals to propagate to the pins, however, ranges from 25 to 125 ns. This propagation delay is dependent on variables such as temperature and pin loading. Propagation delays are approximately 50 ns. The other signals are typically 85 ns. Propagation delays are incorporated in the AC specifications.





This diagram indicates when signals are clocked internally. The time it takes the signals to propagate to the pins, however, ranges from 25 to 125 ns. This propagation delay is dependent on variables such as temperature and pin loading. Propagation also varies from output to output and component to component. Typically though, ( $T_A = 25^\circ\text{C}$ , fully loaded) RD and WR propagation delays are approximately 50 ns. The other signals are typically 85 ns. Propagation delays are incorporated in the AC specifications.

## 8744H EPROM CHARACTERISTICS

### Erase Characteristics

Erase of the 8744H Program Memory begins to occur when the chip is exposed to light with wavelengths shorter than approximately 4,000 Ångströms. Since sunlight and fluorescent lighting have wavelengths in this range, constant exposure to these light sources over an extended period of time (about 1 week in sunlight, or 3 years in room-level fluorescent lighting) could cause unintentional erasure. If an application subjects the 8744H to this type of exposure, it is suggested that an opaque label be placed over the window.

The recommended erasure procedure is exposure to ultraviolet light (at 2537 Ångströms) to an integrated dose of at least 15 W-sec/cm<sup>2</sup> rating for 20 to 30 minutes, at a distance of about 1 inch, should be sufficient.

Erase leaves the array in an all 1s state.

### Programming the EPROM

To be programmed, the 8744H must be running with a 4 to 6 MHz oscillator. (The reason the oscillator needs to be running is that the internal bus is being used to transfer address and program data to appropriate registers.) The address of an EPROM location to be programmed is applied to Port 1 and pins P2.0-P2.3 of Port 2, while the data byte is applied to Port 0. Pins P2.4-P2.6 and PSEN should be held low, and P2.7 and RST high. (These are all TTL levels except RST, which requires 2.5V for high.) EA/VPP is held normally high, and is pulsed to +21V. While EA/VPP is at 21V, the ALE/PROG pin, which is normally being held high, is pulsed low for 50 msec. Then EA/VPP is returned to high. This is illustrated in Figure 3. Detailed timing specifications are provided in the EPROM Programming and Verification Characteristics section of this data sheet.

### Program Memory Security

The program memory security feature is developed around a "security bit" in the 8744H EPROM array. Once this "hidden bit" is programmed, electrical access to the contents of the entire program memory array becomes impossible. Activation of this feature is accomplished by programming the 8744H as described in "Programming the EPROM" with the exception that P2.6 is held at a TTL high rather than a TTL low. In addition, Port 1 and P2.0-P2.3 may be in any state. Figure 4 illustrates the security bit programming configuration. Deactivating the security feature, which again allows programmability of the EPROM, is accomplished by exposing the EPROM to ultraviolet light. This exposure, as described in "Erase Characteristics," erases the entire EPROM array. Therefore, attempted retrieval of "protected code" results in its destruction.

### Program Verification

Program Memory may be read only when the "security feature" has not been activated. Refer to Figure 5 for Program Verification setup. To read the Program Memory, the following procedure can be used. The unit must be running with a 4 to 6 MHz oscillator. The address of a Program Memory location to be read is applied to Port 1 and pins P2.0-P2.3 of Port 2. Pins P2.4-P2.6 and PSEN are held at TTL low, while the ALE/PROG, RST, and EA/VPP pins are held at TTL high. (These are all TTL levels except RST, which requires 2.5V for high.) Port 0 will be the data output lines. P2.7 can be used as a read strobe. While P2.7 is held high, the Port 0 pins float. When P2.7 is strobed low, the contents of the addressed location will appear at Port 0. External pullups (e.g., 10K) are required on Port 0 during program verification.

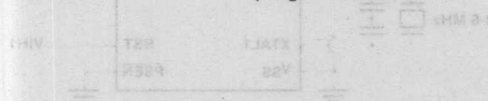


Figure 4 Security Bit Programming Configuration

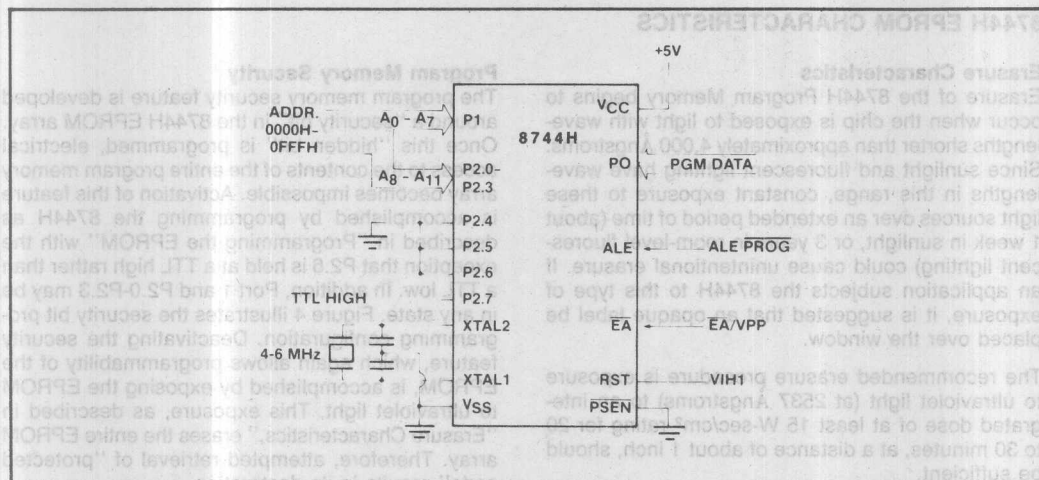


Figure 3. Programming Configuration

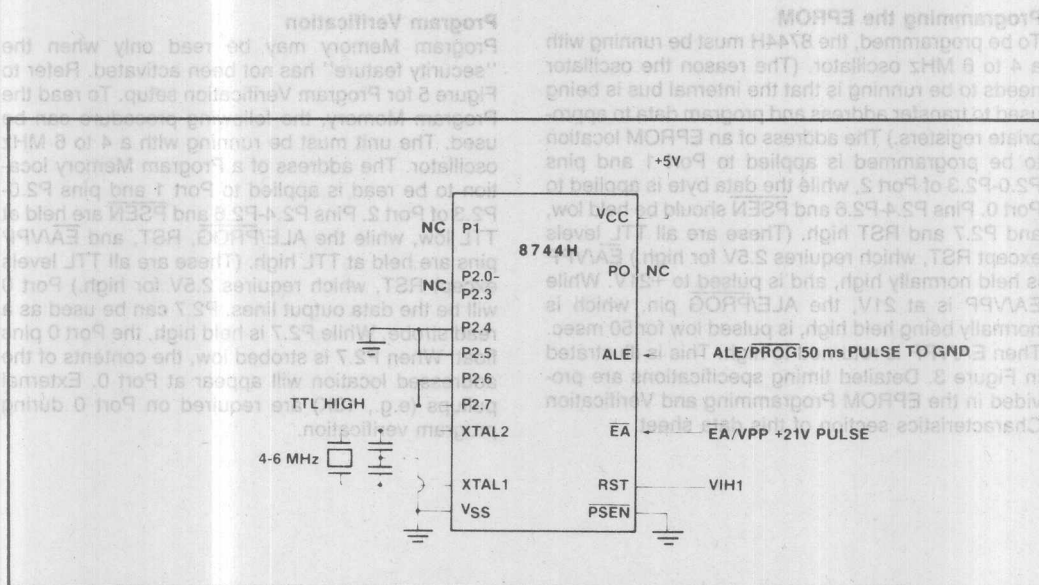


Figure 4. Security Bit Programming Configuration

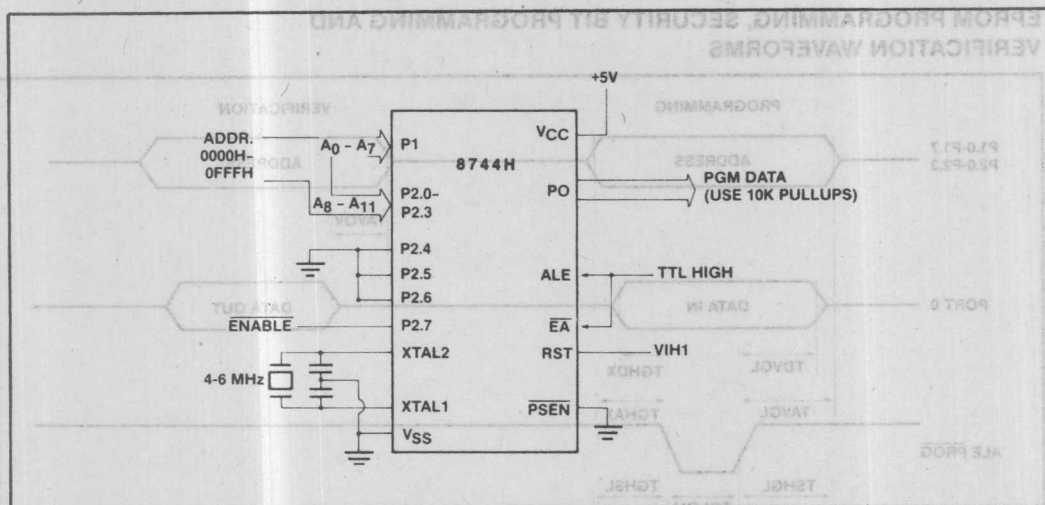


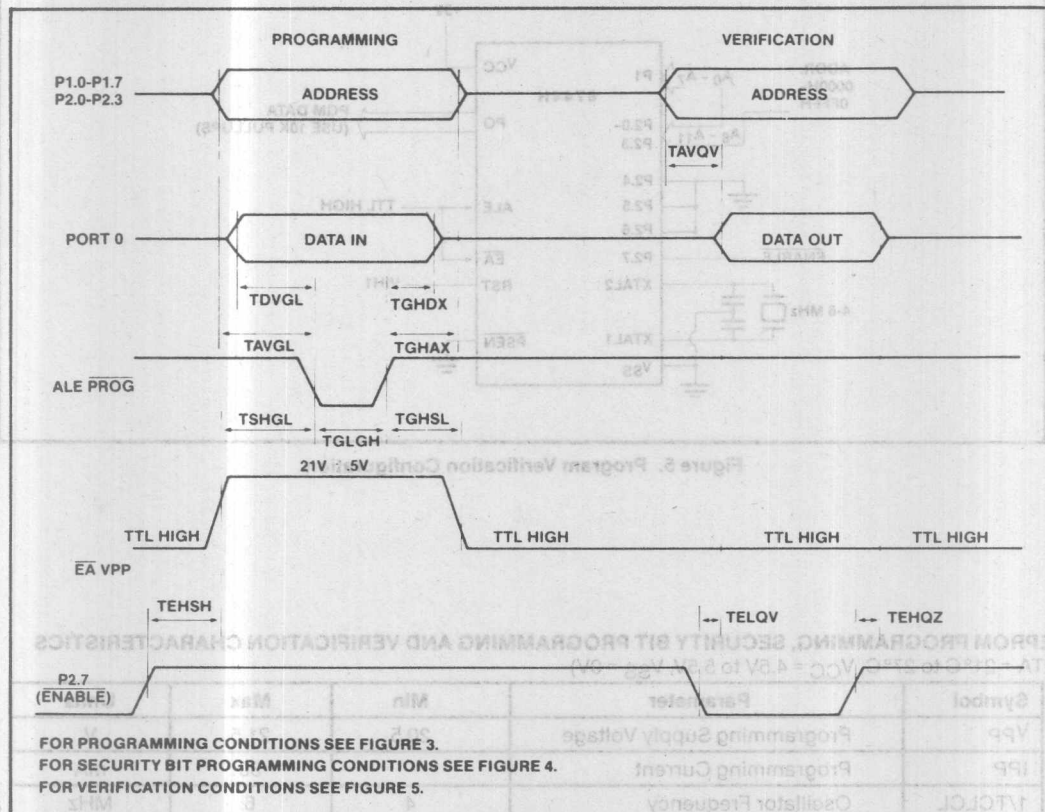
Figure 5. Program Verification Configuration

**EPROM PROGRAMMING, SECURITY BIT PROGRAMMING AND VERIFICATION CHARACTERISTICS**  
(TA = 21°C to 27°C, VCC = 4.5V to 5.5V, VSS = 0V)

Symbol	Parameter	Min	Max	Units
Vpp	Programming Supply Voltage	20.5	21.5	V
IPP	Programming Current		30	mA
1/TCLCL	Oscillator Frequency	4	6	MHz
TAVGL	Address Setup to $\overline{\text{PROG}}$	48TCLCL		
TGHAX	Address Hold after $\overline{\text{PROG}}$	48TCLCL		
TDVGL	Data Setup to $\overline{\text{PROG}}$	48TCLCL		
TGHDX	Data Hold after $\overline{\text{PROG}}$	48TCLCL		
TEHSH	$\overline{\text{ENABLE}}$ High to Vpp	48TCLCL		
TSHGL	Vpp Setup to $\overline{\text{PROG}}$	10		$\mu\text{sec}$
TGHSL	Vpp Hold after $\overline{\text{PROG}}$	10		$\mu\text{sec}$
TGLGH	$\overline{\text{PROG}}$ Width	45	55	msec
TAVQV	Address to Data Valid		48TCLCL	
TELQV	$\overline{\text{ENABLE}}$ to Data Valid		48TCLCL	
TEHQZ	Data Float after $\overline{\text{ENABLE}}$	0	48TCLCL	



# VERIFICATION WAVEFORMS



FOR PROGRAMMING CONDITIONS SEE FIGURE 3.  
 FOR SECURITY BIT PROGRAMMING CONDITIONS SEE FIGURE 4.  
 FOR VERIFICATION CONDITIONS SEE FIGURE 5.

Symbol	Parameter	Min	Max
$T_{AVQV}$	Address Setup to PROG	48TCLCL	
$T_{GHAX}$	Address Hold after PROG	48TCLCL	
$T_{DVGL}$	Data Setup to PROG	48TCLCL	
$T_{GHDX}$	Data Hold after PROG	48TCLCL	
$T_{EHSH}$	ENABLE High to Vpp	48TCLCL	
$T_{SHGL}$	Vpp Setup to PROG	10	48sec
$T_{GHSL}$	Vpp Hold after PROG	10	48sec
$T_{GLGH}$	PROG Width	45	55 msec
$T_{AVOV}$	Address to Data Valid	48TCLCL	
$T_{ELOV}$	ENABLE to Data Valid	48TCLCL	
$T_{EHQZ}$	Data First after ENABLE	0	





ORDER NUMBER 230876-001

# MICROCONTROLLER WITH INTEGRATED HIGH PERFORMANCE COMMUNICATIONS INTERFACE

CHARLES YAGER  
APPLICATIONS ENGINEER

**CHARLES YAGER**  
APPLICATIONS ENGINEER



## SUMMARY

The 8044 offers a lower cost and higher performance solution to networking microcontrollers than conventional solutions. The system cost is lowered by integrating an entire microcomputer with an intelligent HDLC/SDLC communication processor onto a single chip. The higher performance is realized by integrating two processors running concurrently on one chip; the powerful 8051 microcontroller and the Serial Interface Unit. The 8051 microcontroller is substantially off-loaded from the communication tasks when using the AUTO mode. In the AUTO mode the SIU handles many of the data link functions in hardware. The advantages of the AUTO mode are: less software is required to implement a secondary station data link, the 8051 CPU is offloaded, and the turn-around time is reduced, thus increasing the network throughput. Currently the 8044 is the only microcontroller with a sophisticated communications processor on-chip. In the future there will be more microcontrollers available following this trend.

## INTRODUCTION

Today microcontrollers are being designed into virtually every type of equipment. For the household, they are turning up in refrigerators, thermostats, burglar alarms, sprinklers, and even water softeners. At work they are found in laboratory instruments, copiers, elevators, hospital equipment, and telephones. In addition, a lot of microcomputer equipment contains more than one microcontroller. Applications using multiple microcontrollers as well, as the office and home, are now faced with the same requirements that laboratory instruments were faced with 12 years ago — they need to connect them together and have them communicate. This need was satisfied in the laboratory with the IEEE-488 General Purpose Instrumentation Bus (GPIB). However, GPIB does not meet the current design objectives for networking microcontrollers.

Today there are many communications schemes and protocols available; some of the popular ones are GPIB, Async, HDLC/SDLC, and Ethernet. Common design objectives of today's networks are: low cost, reliable, efficient throughput, and expandable. In examining available solutions, GPIB does not meet these design objectives; first, the cable is too expensive (parallel communications), second, it can only be used over a limited distance (20 meters), and third, it can only handle a limited number of stations. For general networking, serial communications is preferable because of lower cable costs and higher reliability (fewer connections). While Ethernet provides very high performance, it is more of a system backbone rather than a microcontroller interconnect. Async, on the other hand, is inexpensive but it is not an efficient protocol for data block or file transfers. Even with some new modifications such as a 9 bit protocol for addressing, important functions such as

acknowledgements, error checking/recovery, and data transparency are not standardized nor supported by available data comm chips.

SDLC, Synchronous Data Link Control, meets the requirements for communications link design. The physical medium can be used on two or four wire twisted pair with inexpensive transceivers and connectors. It can also be interfaced through modems, which allows it to be used on broadband networks, leased or switched telephone lines. VLSI controllers have been available from a number of vendors for years; higher performance and more user friendly SDLC controllers continue to appear. SDLC has also been designed to be very reliable. A 16 bit CRC checks the integrity of the received data, while frame numbering and acknowledgements are also built in. Using SDLC, up to 254 stations can be uniquely addressed, while HDLC addressing is unlimited. If an RS-422 only requires a single +5 volt power supply.

What will the end user pay for the added value provided by communications? The cost of the communications hardware is not the only additional cost. There will be performance degradation in the main application because the microcontroller now has additional tasks to perform. There are two extremes to the cost of adding communication capability. One could spend very little by adding an I/O port and have the CPU handle everything from the baud rate to the protocol. Of course the main application would be idle while the CPU was communicating. The other extreme would be to add another microcontroller to the system dedicated to communications. This communications processor could interface to the main CPU through a high speed parallel link or dual port RAM. This approach would maintain system performance, but it would be costly.

### Adding HDLC/SDLC Networking Capability

Figure 1 shows a microcomputer system with a conventional HDLC/SDLC communications solution. The additional hardware needed to realize the conventional design is: an HDLC/SDLC communication chip, additional ROM for the communication software, part of an interrupt controller, a baud rate generator, a phase locked loop, NRZI encoded/decoder, and a cable driver locked loop are used when the transmitter does not send the clock on a separate line from the data (i.e. over telephone lines, or two wire cable). the NRZI encoder/decoder is used in HDLC/SDLC to combine the clock into the data line. A phase locked loop is used to recover the clock from the data line.

The majority of the available communication chips provide a limited number of data link control functions. Most of them will handle Zero Bit Insertion/Deletion (ZBI/D), Flags, Aborts, Automatic

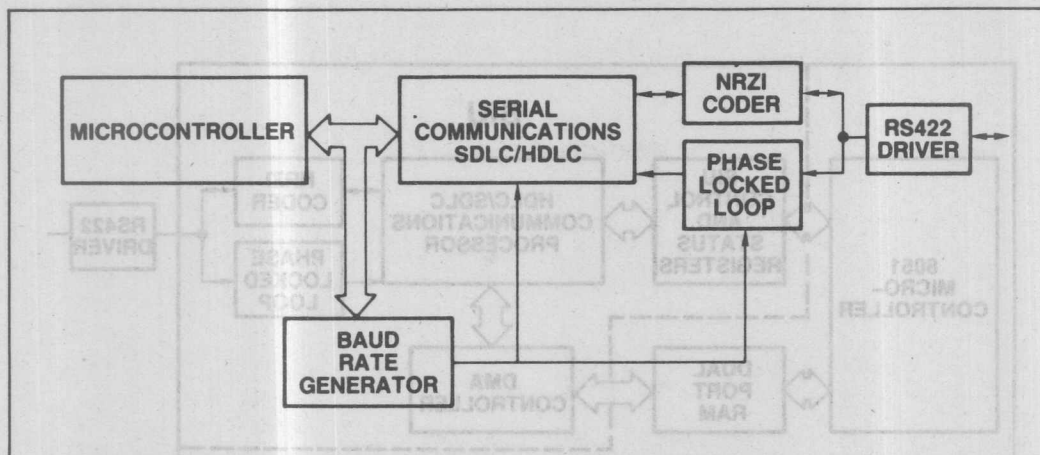


Figure 1. Conventional microcontroller networking solution

address recognition, and CRC generation and checking. It is the CPU's responsibility to manage link access, command recognition and response, acknowledgements and error recovery. Handling these tasks can take a lot of CPU time. In addition, servicing the transmission and reception of data bytes can also be very time consuming depending on the method used.

Using a DMA controller can increase the overall system performance, since it can transfer a block of data in fewer clock cycles than a CPU. In addition, the CPU and the DMA controller can multiplex their access to the bus so that both can be running at virtually the same time. However, both the DMA controller and the CPU are sharing the same bus, therefore, neither one get to utilize 100% of the bus bandwidth. Microcontrollers available today do not support DMA, therefore, they would have to use interrupts, since polling is unacceptable in a multitasking environment.

In an interrupt driven, the CPU has overhead in addition to servicing the interrupt. During each interrupt request the CPU has to save all of the important registers, transfer a byte, update pointers and counters, then restore all of its registers. At low bit rates this overhead may be insignificant. However, the percentage of overhead increases linearly with the bit rate. At high bit rates this overhead would consume all of the CPU's time. There is another nuisance factor associated with interrupt driven systems, interrupt latency. Too much interrupt latency will cause data to be lost from underrun and overrun errors.

The additional hardware necessary to implement the communications solution, as shown in Figure 1, would

require 1 LSI chip and about 10 TTL chips. The cost of CPU throughput degradation can be even greater. The percentage of time the CPU has to spend servicing the communication tasks can be anywhere from 10-100%, depending on the serial bit rate. These high costs will prevent consumer acceptance of networking microcomputer equipment.

#### A Highly Integrated, High Performance Solution

The 8044 reduces the cost of networking microcontrollers without compromising performance. It contains all of the hardware components necessary to implement a microcomputer system with communications capability, plus it reduces the CPU and software overhead of implementing HDLC/SDLC. Figure 2 shows a functional block diagram of the 8044.

The 8044 integrates the powerful 8051 microcontroller with an intelligent Serial Interface Unit to provide a single chip solution which efficiently implements a distributed processing or distributed control system. The microcontroller is a self sufficient unit containing ROM, RAM, ALU and its own peripherals. The 8044's architecture and instruction set are identical to the 8051's. The Serial Interface Unit (SIU) uses bit synchronous HDLC/SDLC protocol and can communicate at bit rates up to 2.4 Mbps, externally clocked, or up to 375 Kbps using the on-chip digital phase locked loop. The SIU contains its own processor, which operates concurrently with the microcontroller.

The CPU and the SIU, in the 8044, interface through 192 bytes of dual port RAM. There is no hardware arbitration in the dual port RAM. Both processor's memory access cycles are interlaced; each processor has access every other clock cycle. Therefore, there

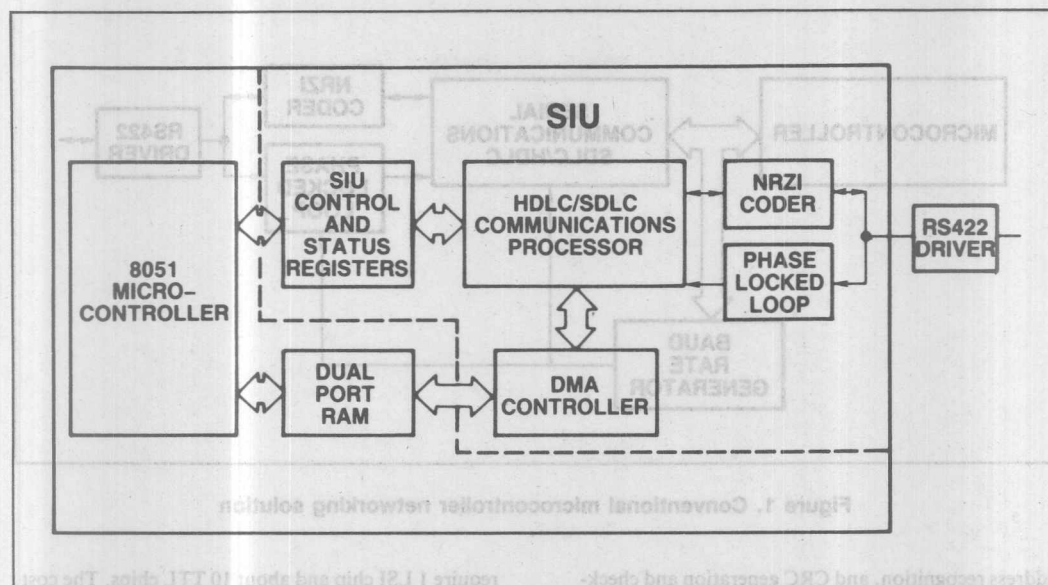


Figure 2. 8044 single chip microcontroller networking solution

is no throughput loss in either processor as a result of the dual port RAM, and execution times are deterministic. Since this has always been the method for memory access on the 8051 microcontroller, 8051 programs have the same execution time in the 8044.

By integrating all of the communication hardware onto the 8051 microcontroller, the hardware cost of the system is reduced. Now several chips have been integrated into a single chip. This means that the system power is reduced, P.C. board space is reduced, inventory and assembly is reduced, and reliability is improved. The improvement in reliability is a result of fewer chips and interconnections on the P.C. board.

As mentioned before, there can be two extremes in a design which adds communications to the microcomputer system. The 8044 solution uses the high end extreme. The SIU on the 8044 contains its own processor which communicates with the 8051 processor through dual port RAM and control/status registers. While the SIU is not a totally independent communications processor, it substantially offloads the 8051 processor from the communication tasks.

The DMA on the 8044 is dedicated to the SIU. It cannot access external RAM. By having a DMA controller in the SIU, the 8051 CPU is offloaded. As a result of the dual port RAM design, the DMA does not share the running at full speed while the frames are being

transmitted or received. Also, the nuisance of overrun and underrun errors is totally eliminated since the dedicated DMA controller is guaranteed to meet the maximum data rates. Having a dedicated DMA controller means that the serial channel interrupt can be the lowest priority, thus allowing the CPU to have higher priority real time interrupts.

Figure 3 shows a comparison between the conventional and the 8044 solution on the percentage of time the CPU must spend sending data. This diagram was derived by assuming a 64 byte information frame is being transmitted repeatedly. The conventional solution is interrupt driven, and each interrupt service routine is assumed to take about 15 instructions with a 1  $\mu$ sec instruction cycle time. At 533 Kbps, an interrupt would occur every 15  $\mu$ sec. Thus, the CPU becomes completely dedicated to servicing the serial communications. The conventional design could not support bit rates higher than this because of underruns and overruns. For the 8044 to repeatedly send 64 byte frames, it simply has to reinitialize the DMA controller. As a result, the 8044 can support bit rates up to 2.4 Mbps.

Some of the other communications tasks the CPU has to perform, such as link access, command recognition/response, and acknowledgements, are performed automatically by the SIU in a mode called "AUTO." The combination of the dedicated DMA controller and the AUTO mode, substantially offload



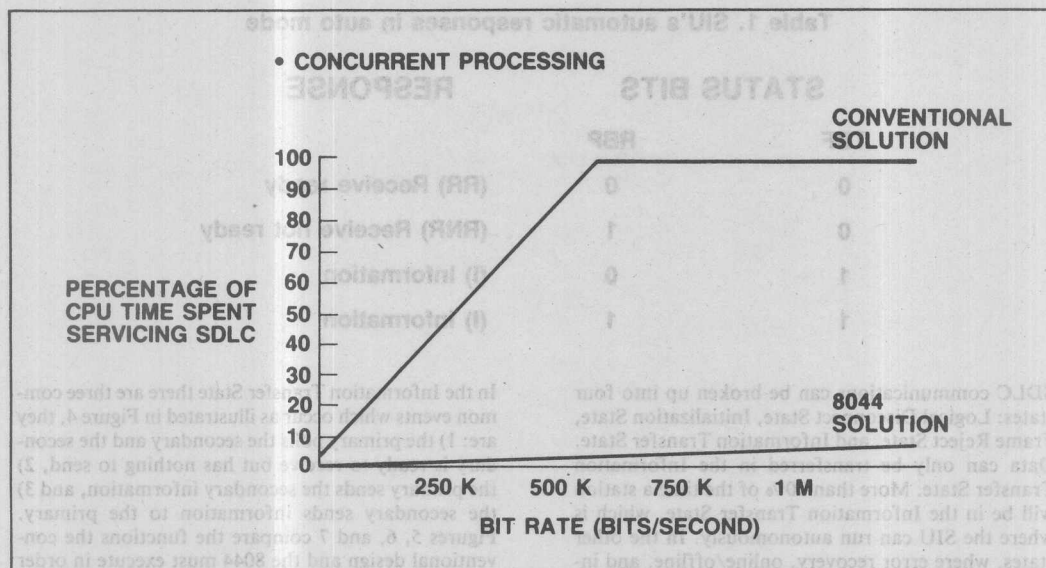


Figure 3. SIU offloads CPU

the CPU, thus allowing it to devote more of its power to other tasks.

#### 8044's Auto Mode

In the AUTO mode the SIU implements in hardware a subset of the SDLC protocol such that it responds to many SDLC commands without CPU intervention. All AUTO mode responses to the primary station conform to IBM's SDLC definition. In the AUTO mode the 8044 can only be a secondary station operating in SDLC specified "Normal Response Mode." Normal Response Mode means that the secondary station can not transmit unless it is polled by the primary station. The SIU in the AUTO mode can recognize and respond to the following SDLC commands without CPU intervention: I (Information), RR (Receive Ready), RNR (Receive Not Ready), REJ (Reject), and for loop mode UP (Unnumbered Poll). The SIU can generate the following responses without CPU intervention: I, RR, and RNR. In addition, the SIU manages Ns and Nr in the control field. If it detects an error in either Ns or Nr, it interrupts the CPU for error recovery.

How does the SIU know what responses to send to the primary? It uses two status bits which are set by the CPU. The two bits are TBF (Transmit Buffer Full) and RBP (Receive Buffer Protect). TBF indicates that the CPU wants to send data, and RBP indicates that the receive data buffer is full. Table 1 shows the responses the SIU will send based on these two status bits. This is an innovative approach to communication design. The CPU in the 8044 with one instruction

can directly set a bit which communicates to the primary what its transmit and receive buffering status is.

When the CPU wants to send a frame, it loads the transmit buffer with the data, loads the starting address and the count of the data into the SIU, then sets TBF to transmit the frame. The SIU waits for the primary station to poll it with a RR command. After the SIU is polled, it automatically sends the information frame to the primary with the proper control field. The SIU then waits for a positive acknowledgement from the primary before incrementing the Ns field and interrupting the CPU for more data. If a negative acknowledgement is received, the SIU automatically retransmits the frame.

When the 8044 is ready to receive information, the CPU loads the receive buffer starting address and the buffer length into the SIU, then enables the receiver. When a valid information frame with the correct address and CRC is received, the SIU will increment the Nr field, disable the receiver and interrupt the CPU indicating that a good I frame has been received. The CPU then sets RBP, reenables the receiver and processes the received data. By enabling the receiver with RBP set, the SIU will automatically respond to polls with a Receive Not Ready, thus keeping the link moving rather than timing out the primary from a disabled receiver, or interrupting the CPU with another poll before it has processed the data. After the data has been processed, the CPU clears RBP, returning to the Receive Ready responses.



## STATUS BITS

## RESPONSE

TBF

RBP

0

0

(RR) Receive ready

0

1

(RNR) Receive not ready

1

0

(I) Information

1

1

(I) Information

SDLC communications can be broken up into four states: Logical Disconnect State, Initialization State, Frame Reject State, and Information Transfer State. Data can only be transferred in the Information Transfer State. More than 90% of the time a station will be in the Information Transfer State, which is where the SIU can run autonomously. In the other states, where error recovery, online/offline, and initialization takes place, the CPU manages the protocol.

In the Information Transfer State there are three common events which occur as illustrated in Figure 4, they are: 1) the primary polls the secondary and the secondary is ready to receive but has nothing to send, 2) the primary sends the secondary information, and 3) the secondary sends information to the primary. Figures 5, 6, and 7 compare the functions the conventional design and the 8044 must execute in order to respond to the primary for the cases in Figure 4.

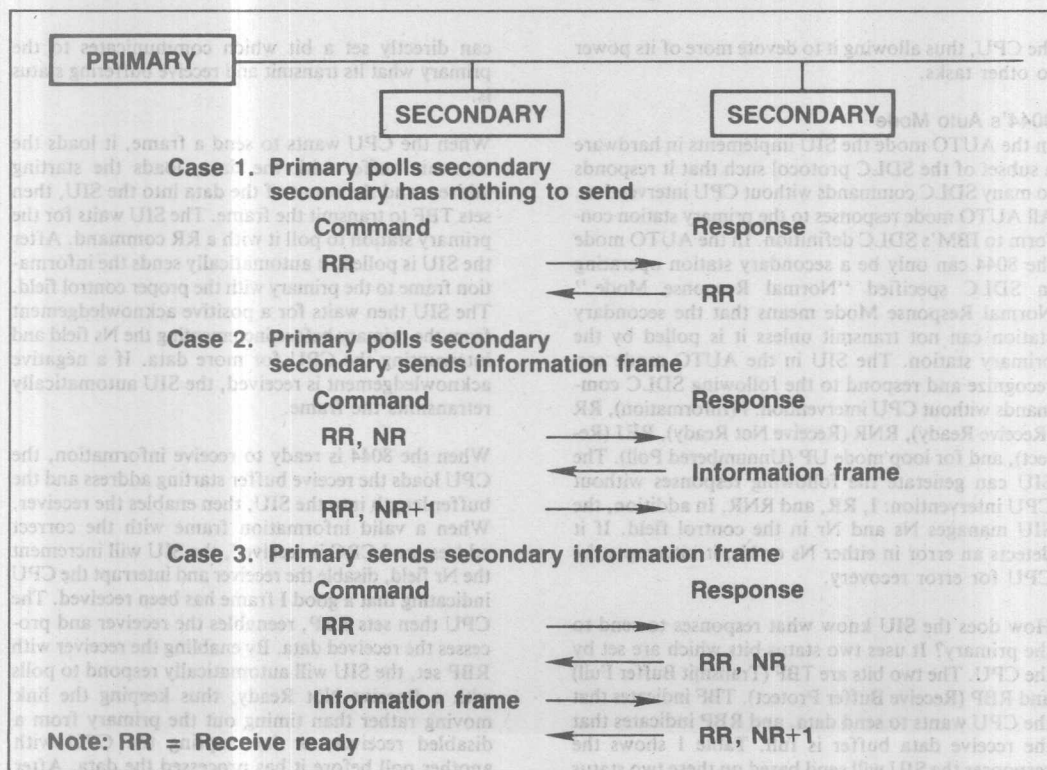


Figure 4. SDLC commands and responses in the information transfer state

## CASE 1

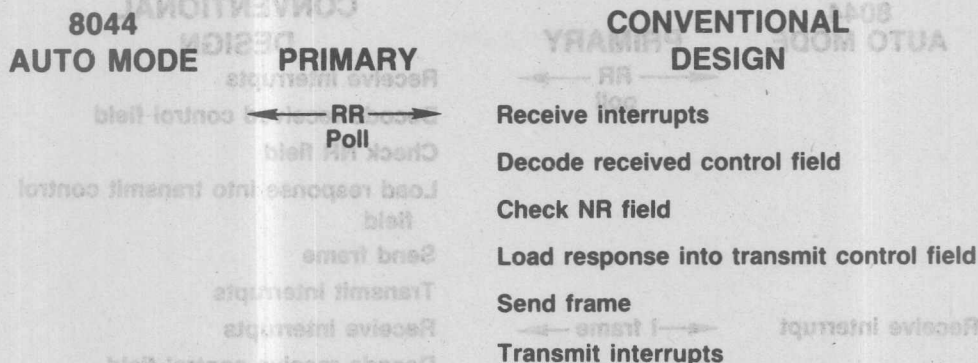


Figure 5. Primary polls secondary, secondary has nothing to send

## CASE 2

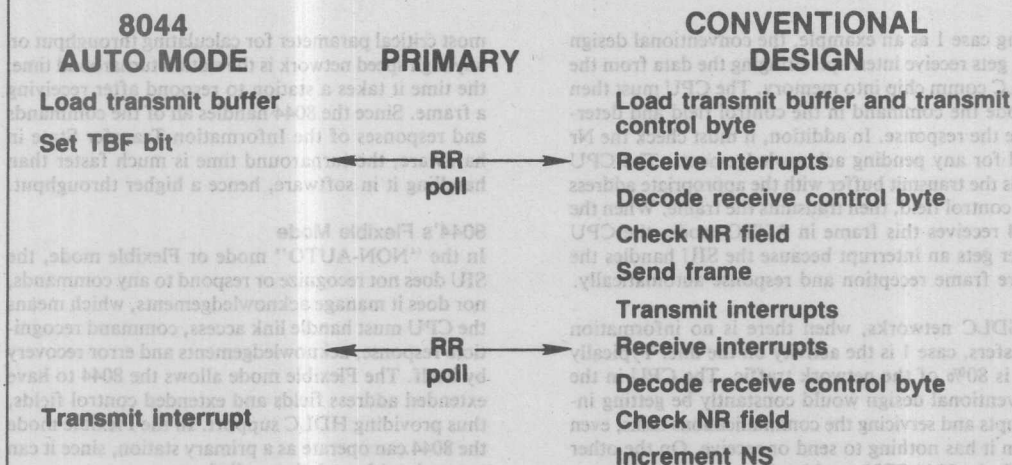
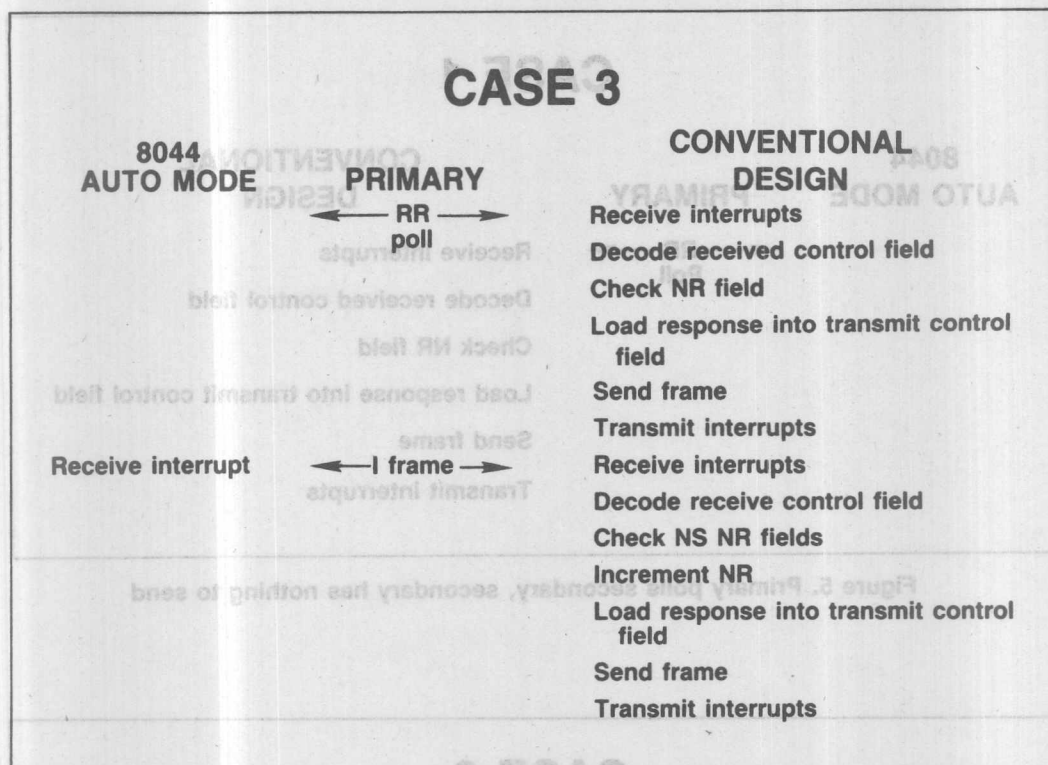


Figure 6. Primary polls secondary, secondary sends information frame



**Figure 7. Primary sends information frame to secondary**

Using case 1 as an example, the conventional design first gets receive interrupts bringing the data from the SDLC comm chip into memory. The CPU must then decode the command in the control field and determine the response. In addition, it must check the Nr field for any pending acknowledgements. The CPU loads the transmit buffer with the appropriate address and control field, then transmits the frame. When the 8044 receives this frame in AUTO mode, the CPU never gets an interrupt because the SIU handles the entire frame reception and response automatically.

In SDLC networks, when there is no information transfers, case 1 is the activity on the line. Typically this is 80% of the network traffic. The CPU in the conventional design would constantly be getting interrupts and servicing the communications tasks, even when it has nothing to send or receive. On the other hand, the 8044 CPU would only get involved in communicating when it has data to send or receive.

Having the SIU implement a subset of the SDLC protocol in hardware not only offloads the CPU, but it also improves the throughput on the network. The

most critical parameter for calculating throughput on any high speed network is the station turnaround time; the time it takes a station to respond after receiving a frame. Since the 8044 handles all of the commands and responses of the Information Transfer State in hardware, the turnaround time is much faster than handling it in software, hence a higher throughput.

#### 8044's Flexible Mode

In the "NON-AUTO" mode or Flexible mode, the SIU does not recognize or respond to any commands, nor does it manage acknowledgements, which means the CPU must handle link access, command recognition/response, acknowledgements and error recovery by itself. The Flexible mode allows the 8044 to have extended address fields and extended control fields, thus providing HDLC support. In the Flexible mode the 8044 can operate as a primary station, since it can transmit without being polled.

#### Front End Communications Processor

The 8044 can also be used as an intelligent HDLC/SDLC front end for a microprocessor, capable of extensively off-loading link control functions for

the microprocessor. In some applications the 8044 can even be used for communications preprocessing, in addition to data link control. For this type of design the 8044 would communicate to the Host CPU through a FIFO, or dual port RAM. A block diagram of this design is given in Figure 8. A tightly coupled interface between the 8044 and the Host CPU would be established. The Host CPU would give the 8044 high level commands and data which the 8044 would convert to HDLC/SDLC. This particular type of design would be most appropriate for a primary Station which is normally a micro, mini, or mainframe

computer. Sophisticated secondary stations could also take advantage of this design.

Since the 8044 has ROM on chip, all the communications software is non-volatile. The 8044 primary station could down-line-load software to 8044 secondary stations. Once down-line-loading is implemented, software updates to the primary and secondary stations could be done very inexpensively. The only things which would remain fixed in ROM are the HDLC/SDLC communications software and the software interface to the HOST.

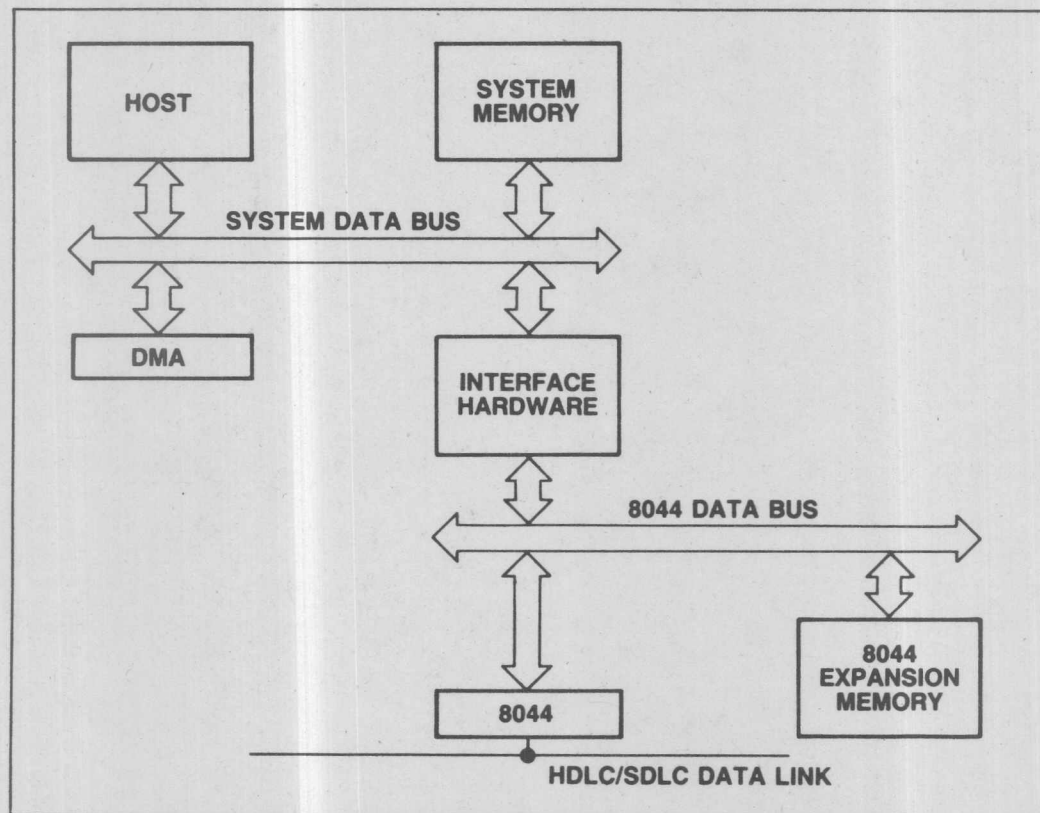


Figure 8. 8044 front end processor



even be used for communications preprocessing, in addition to data link control. For this type of design the 8044 would communicate to the Host CPU through a FIFO or dual port RAM. A block diagram of this design is given in Figure 8. A tightly coupled interface between the 8044 and the Host CPU would be established. The Host CPU would give the 8044 high level commands and data which the 8044 would convert to HDLC/SDLC. This particular type of design would be most appropriate for a primary station which is normally a micro, mini, or mainframe.

Since the 8044 has ROM on chip, all the communications software is non-volatile. The 8044 primary station could download software to 8044 secondary stations. Once download is implemented, software updates to the primary and secondary stations could be done very inexpensively. The only things which would remain fixed in ROM are the HDLC/SDLC communications software and the software interface to the HOST.

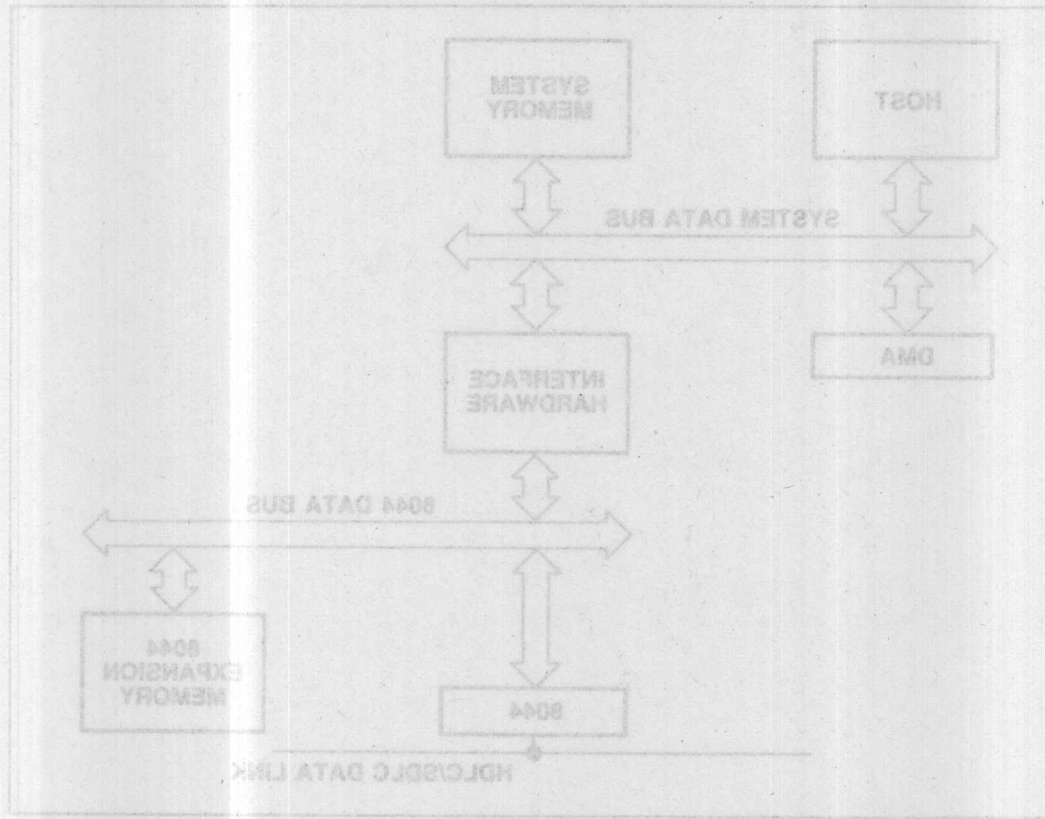


Figure 8. 8044 front end processor





# Designing Microcontroller Systems for Electrically Noisy Environments

## Contents

<b>SYMPTOMS OF NOISE PROBLEMS</b>	22-2
<b>TYPES AND SOURCES OF ELECTRICAL NOISE</b>	
Supply Line Transients	22-2
EMP and RFI	22-2
ESD	22-3
Ground Noise	22-3
<b>"RADIATED" AND "CONDUCTED" NOISE</b>	22-3
<b>SIMULATING THE ENVIRONMENT</b>	22-4
<b>TYPES OF FAILURES AND FAILURE MECHANISMS</b>	22-4
<b>THE GAME PLAN</b>	22-5
<b>CURRENT LOOPS</b>	22-5
<b>SHIELDING</b>	22-6
Shielding Against Capacitive Coupling	22-6
Shielding Against Inductive Coupling	22-6
RF Shielding	22-9
<b>GROUNDING</b>	22-10
Safety Ground	22-10
Signal Ground	22-11
Practical Grounding	22-12
Braided Cable	22-13
<b>POWER SUPPLY DISTRIBUTION AND DECOUPLING</b>	22-14
Selecting the Value of the Decoupling Cap	22-15
The Case for On-Board Voltage Regulation	22-16
<b>RECOVERING GRACEFULLY FROM A SOFTWARE UPSET</b>	22-16
<b>SPECIAL PROBLEM AREAS</b>	22-18
ESD	22-18
The Automotive Environment	22-19
<b>PARTING THOUGHTS</b>	22-21
<b>REFERENCES</b>	22-22

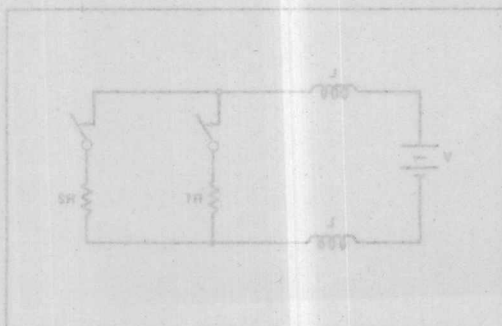


Figure 1: Supply Line Transients



Digital circuits are often thought of as being immune to noise problems, but really they're not. Noises in digital systems produce software upsets: program jumps to apparently random locations in memory. Noise-induced glitches in the signal lines can cause such problems, but the supply voltage is more sensitive to glitches than the signal lines.

Severe noise conditions, those involving electrostatic discharges, or as found in automotive environments, can do permanent damage to the hardware. Electrostatic discharges can blow a crater in the silicon. In the automotive environment, in ordinary operation, the "12V" power line can show + and -400V transients.

This Application Note describes some electrical noises and noise environments. Design considerations, along the lines of PCB layout, power supply distribution and decoupling, and shielding and grounding techniques, that may help minimize noise susceptibility are reviewed. Special attention is given to the automotive and ESD environments.

## Symptoms of Noise Problems

Noise problems are not usually encountered during the development phase of a microcontroller system. This is because benches rarely simulate the system's intended environment. Noise problems tend not to show up until the system is installed and operating in its intended environment. Then, after a few minutes or hours of normal operation the system finds itself someplace out in left field. Inputs are ignored and outputs are gibberish. The system may respond to a reset, or it may have to be turned off physically and then back on again, at which point it commences operating as though nothing had happened. There may be an obvious cause, such as an electrostatic discharge from somebody's finger to a keyboard or the upset occurs every time a copier machine is turned on or off. Or there may be no obvious cause, and nothing the operator can do will make the upset repeat itself. But a few minutes, or a few hours, or a few days later it happens again.

One symptom of electrical noise problems is randomness, both in the occurrence of the problem and in what the system does in its failure. All operational upsets that occur at seemingly random intervals are not necessarily caused by noise in the system. Marginal VCC, inadequate decoupling, rarely encountered software conditions, or timing coincidences can produce upsets that seem to occur randomly. On the other hand, some noise sources can produce upsets downright periodically. Nevertheless, the more difficult it is to characterize an upset as to cause and effect, the more likely it is to be a noise problem.

## Types and Sources of Electrical Noise

The name given to electrical noises other than those that are inherent in the circuit components (such as thermal noise) is EMI: electromagnetic interference. Motors, power switches, fluorescent lights, electrostatic discharges, etc., are sources of EMI. There is a veritable alphabet soup of EMI types, and these are briefly described below.

### SUPPLY LINE TRANSIENTS

Anything that switches heavy current loads onto or off of AC or DC power lines will cause large transients in these power lines. Switching an electric typewriter on or off, for example, can put a 1000V spike onto the AC power lines.

The basic mechanism behind supply line transients is shown in Figure 1. The battery represents any power source, AC or DC. The coils represent the line inductance between the power source and the switchable loads R1 and R2. If both loads are drawing current, the line current flowing through the line inductance establishes a magnetic field of some value. Then, when one of the loads is switched off, the field due to that component of the line current collapses, generating transient voltages,  $v = L(di/dt)$ , which try to maintain the current at its original level. That's called an "inductive kick." Because of contact bounce, transients are generated whether the switch is being opened or closed, but they're worse when the switch is being opened.

An inductive kick of one type or another is involved in most line transients, including those found in the automotive environment. Other mechanisms for line transients exist, involving noise pickup on the lines. The noise voltages are then conducted to a susceptible circuit right along with the power.

### EMP AND RFI

Anything that produces arcs or sparks will radiate electromagnetic pulses (EMP) or radio-frequency interference (RFI).

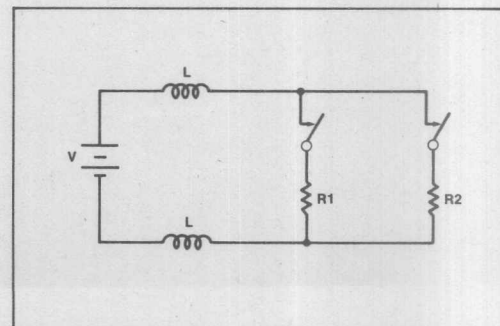


Figure 1. Supply Line Transients

Spark discharges have probably caused more software upsets in digital equipment than any other single noise source. The upsetting mechanism is the EMP produced by the spark. The EMP induces transients in the circuit, which are what actually cause the upset.

Arcs and sparks occur in automotive ignition systems, electric motors, switches, static discharges, etc. Electric motors that have commutator bars produce an arc as the brushes pass from one bar to the next. DC motors and the "universal" (AC/DC) motors that are used to power hand tools are the kinds that have commutator bars. In switches, the same inductive kick that puts transients on the supply lines will cause an opening or closing switch to throw a spark.

### ESD

Electrostatic discharge (ESD) is the spark that occurs when a person picks up a static charge from walking across a carpet, and then discharges it into a keyboard, or whatever else can be touched. Walking across a carpet in a dry climate, a person can accumulate a static voltage of 35kV. The current pulse from an electrostatic discharge has an extremely fast risetime — typically, 4A/nsec. Figure 2 shows ESD waveforms that have been observed by some investigators of ESD phenomena.

It is enlightening to calculate the  $L(di/dt)$  voltage required to drive an ESD current pulse through a couple of inches of straight wire. Two inches of straight wire has about 50nH of inductance. That's not very much, but using 50nH for L and 4A/nsec for  $di/dt$  gives an  $L(di/dt)$  drop of about 200V. Recent observations by W.M. King suggest even faster risetimes (Figure 2B) and the occurrence of multiple discharges during a single discharge event.

Obviously, ESD-sensitivity needs to be considered in the design of equipment that is going to be subjected to it, such as office equipment.

### GROUND NOISE

Currents in ground lines are another source of noise. These can be 60Hz currents from the power lines, or RF hash, or crosstalk from other signals that are sharing this particular wire as a signal return line. Noise in the ground lines is often referred to as a "ground loop" problem. The basic concept of the ground loop is shown in Figure 3. The problem is that true earth-ground is not really at the same potential in all locations. If the two ends of a wire are earth-grounded at different locations, the voltage difference between the two "ground" points can drive significant currents (several amperes) through the wire. Consider the wire to be part of a loop which contains, in addition to the wire, a voltage source that represents the difference in potential between the two ground points, and you have the classical "ground loop." By extension, the term is used to refer to any unwanted (and often unexpected) currents in a ground line.

### "Radiated" and "Conducted" Noise

Radiated noise is noise that arrives at the victim circuit in the form of electromagnetic radiation, such as EMP and RFI. It causes trouble by inducing extraneous voltages in the circuit. Conducted noise is noise that arrives at the victim circuit already in the form of an extraneous voltage, typically via the AC or DC power lines.

One defends against radiated noise by care in designing layouts and the use of effective shielding techniques. One defends against conducted noise with filters and suppres-

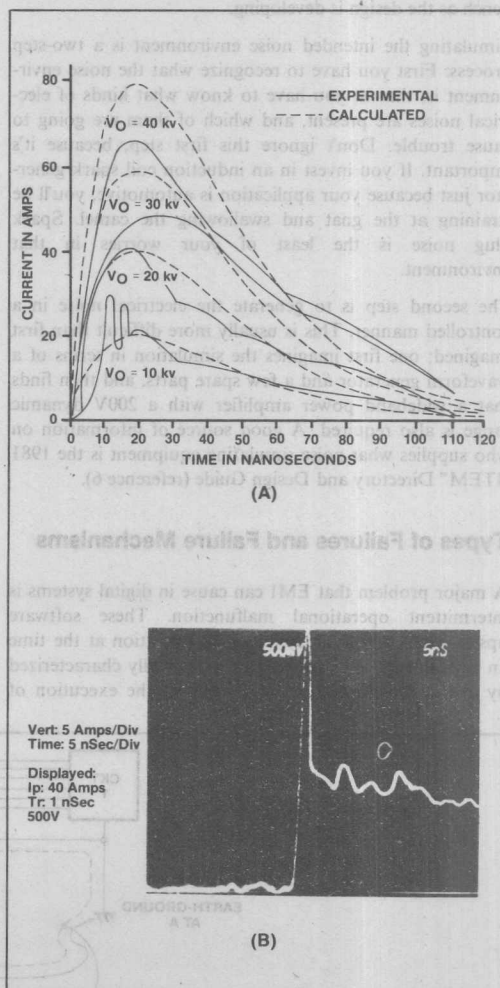


Figure 2. Waveforms of Electrostatic Discharge Currents From a Hand-Held Metallic Object

sors, although layouts and grounding techniques are important here, too.

## Simulating the Environment

Addressing noise problems after the design of a system has been completed is an expensive proposition. The ill will generated by failures in the field is not cheap either. It's cheaper in the long run to invest a little time and money in learning about noise and noise simulation equipment, so that controlled tests can be made on the bench as the design is developing.

Simulating the intended noise environment is a two-step process: First you have to recognize what the noise environment is, that is, you have to know what kinds of electrical noises are present, and which of them are going to cause trouble. Don't ignore this first step, because it's important. If you invest in an induction coil spark generator just because your application is automotive, you'll be straining at the gnat and swallowing the camel. Spark plug noise is the least of your worries in that environment.

The second step is to generate the electrical noise in a controlled manner. This is usually more difficult than first imagined; one first imagines the simulation in terms of a waveform generator and a few spare parts, and then finds that a wideband power amplifier with a 200V dynamic range is also required. A good source of information on who supplies what noise-simulating equipment is the 1981 "ITEM" Directory and Design Guide (reference 6).

## Types of Failures and Failure Mechanisms

A major problem that EMI can cause in digital systems is intermittent operational malfunction. These software upsets occur when the system is in operation at the time an EMI source is activated, and are usually characterized by a loss of information or a jump in the execution of

the program to some random location in memory. The person who has to iron out such problems is tempted to say the program counter went crazy. There is usually no damage to the hardware, and normal operation can resume as soon as the EMI has passed or the source is de-activated. Resuming normal operation usually requires manual or automatic reset, and possibly re-entering of lost information.

Electrostatic discharges from operating personnel can cause not only software upsets, but also permanent ("hard") damage to the system. For this to happen the system doesn't even have to be in operation. Sometimes the permanent damage is latent, meaning the initial damage may be marginal and require further aggravation through operating stress and time before permanent failure takes place. Sometimes too the damage is hidden.

One ESD-related failure mechanism that has been identified has to do with the bias voltage on the substrate of the chip. On some CPU chips the substrate is held at -2.5V by a phase-shift oscillator working into a capacitor/diode clamping circuit. This is called a "charge pump" in chip-design circles. If the substrate wanders too far in either direction, program read errors are noted. Some designs have been known to allow electrostatic discharge currents to flow directly into port pins of an 8048. The resulting damage to the oxide causes an increase in leakage current, which loads down the charge pump, reducing the substrate voltage to a marginal or unacceptable level. The system is then unreliable or completely inoperative until the CPU chip is replaced. But if the CPU chip was subjected to a discharge spark once, it will eventually happen again.

Chips that have a grounded substrate, such as the 8748, can sometimes sustain some oxide damage without actually becoming inoperative. In this case the damage is present, and the increased leakage current is noted; however, since the substrate voltage retains its design value, the damage is largely hidden.

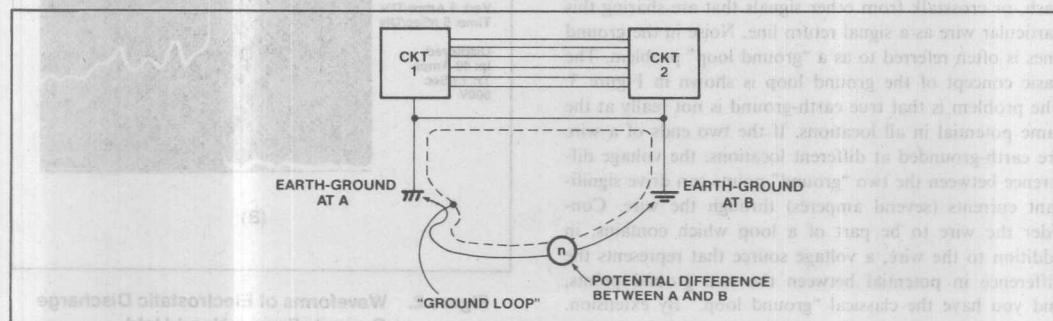


Figure 3. What a Ground Loop Is

It must therefore be recognized that connecting port pins unprotected to a keyboard or to anything else that is subject to electrostatic discharges, makes an extremely dangerous configuration. It doesn't make any difference what CPU chip is being used, or who makes it. If it connects unprotected to a keyboard, it will eventually be destroyed. Designing for an ESD-environment will be discussed further on.

We might note here that MOS chips are not the only components that are susceptible to permanent ESD damage. Bipolar and linear chips can also be damaged in this way. PN junctions are subject to a hard failure mechanism called thermal secondary breakdown, in which a current spike, such as from an electrostatic discharge, causes microscopically localized spots in the junction to approach melt temperatures. Low power TTL chips are subject to this type of damage, as are op-amps. Op-amps, in addition, often carry on-chip MOS capacitors which are directly across an external pin combination, and these are susceptible to dielectric breakdown.

We return now to the subject of software upsets. Noise transients can upset the chip through any pin, even an output pin, because every pin on the chip connects to the substrate through a pn junction. However, the most vulnerable pin is probably the VCC line, since it has direct access to all parts of the chip: every register, gate, flip-flop and buffer.

The menu of possible upset mechanisms is quite lengthy. A transient on the substrate at the wrong time will generally cause a program read error. A false level at a control input can cause an extraneous or misdirected opcode fetch. A disturbance on the supply line can flip a bit in the program counter or instruction register. A short interruption or reversal of polarity on the supply line can actually turn the processor off, but not long enough for the power-up reset capacitor to discharge. Thus when the transient ends, the chip starts up again without a reset.

A common failure mode is for the processor to lock itself into a tight loop. Here it may be executing the data in a table, or the program counter may have jumped a notch, so that the processor is now executing operands instead of opcodes, or it may be trying to fetch opcodes from a nonexistent external program memory.

It should be emphasized that mechanisms for upsets have to do with the arrival of noise-induced transients at the pins of the chips, rather than with the generation of noise pulses within the chip itself, that is, it's not the chip that is picking up noise, it's the circuit.

## The Game Plan

Prevention is usually cheaper than suppression, so first we'll consider some preventive methods that might help to

minimize the generation of noise voltages in the circuit. These methods involve grounding, shielding, and wiring techniques that are directed toward the mechanisms by which noise voltages are generated in the circuit. We'll also discuss methods of decoupling. Then we'll look at some schemes for making a graceful recovery from upsets that occur in spite of preventive measures. Lastly, we'll take another look at two special problem areas: electrostatic discharges and the automotive environment.

## Current Loops

The first thing most people learn about electricity is that current won't flow unless it can flow in a closed loop. This simple fact is sometimes temporarily forgotten by the overworked engineer who has spent the past several years mastering the intricacies of the DO loop, the timing loop, the feedback loop, and maybe even the ground loop. The simple current loop probably owes its apparent demise to the invention of the ground symbol. By a stroke of the pen one avoids having to draw the return paths of most of the current loops in the circuit. Then "ground" turns into an infinite current sink, so that any current that flows into it is gone and forgotten. Forgotten it may be, but it's not gone. It must return to its source, so that its path will by all the laws of nature form a closed loop.

The physical geometry of a given current loop is the key to why it generates EMI, why it's susceptible to EMI, and how to shield it. Specifically, it's the area of the loop that matters.

Any flow of current generates a magnetic field whose intensity varies inversely to the distance from the wire that carries the current. Two parallel wires conducting currents  $+I$  and  $-I$  (as in signal feed and return lines) would generate a nonzero magnetic field near the wires, where the distance from a given point to one wire is noticeably different from the distance to the other wire, but farther away (relative to the wire spacing), where the distances from a given point to either wire are about the same, the fields from both wires tend to cancel out. Thus, maintaining proximity between feed and return paths is an important way to minimize their interference with other signals. The way to maintain their proximity is essentially to minimize their loop area. And, because the mutual inductance from current loop A to current loop B is the same as the mutual inductance from current loop B to current loop A, a circuit that doesn't radiate interference doesn't receive it either.

Thus, from the standpoint of reducing both generation of EMI and susceptibility to EMI, the hard rule is to keep loop areas small. To say that loop areas should be minimized is the same as saying the circuit inductance should



be minimized. Inductance is by definition the constant of proportionality between current and the magnetic field it produces:  $\phi = LI$ . Holding the feed and return wires close together so as to promote field cancellation can be described either as minimizing the loop area or as minimizing  $L$ . It's the same thing.

## Shielding

There are three basic kinds of shields: shielding against capacitive coupling, shielding against inductive coupling, and RF shielding. Capacitive coupling is electric field coupling, so shielding against it amounts to shielding against electric fields. As will be seen, this is relatively easy. Inductive coupling is magnetic field coupling, so shielding against it is shielding against magnetic fields. This is a little more difficult. Strangely enough, this type of shielding does not in general involve the use of magnetic materials. RF shielding, the classical "metallic barrier" against all sorts of electromagnetic fields, is what most people picture when they think about shielding. Its effectiveness depends partly on the selection of the shielding material, but mostly, as it turns out, on the treatment of its seams and the geometry of its openings.

### SHIELDING AGAINST CAPACITIVE COUPLING

Capacitive coupling involves the passage of interfering signals through mutual or stray capacitances that aren't shown on the circuit diagram, but which the experienced engineer knows are there. Capacitive coupling to one's body is what would cause an unstable oscillator to change its frequency when the person reaches his hand over the circuit, for example. More importantly, in a digital system it causes crosstalk in multi-wire cables.

The way to block capacitive coupling is to enclose the circuit or conductor you want to protect in a metal shield. That's called an electrostatic or Faraday shield. If coverage is 100%, the shield does not have to be grounded, but it usually is, to ensure that circuit-to-shield capacitances go to signal reference ground rather than act as feedback and crosstalk elements. Besides, from a mechanical point of view, grounding it is almost inevitable.

A grounded Faraday shield can be used to break capacitive coupling between a noisy circuit and a victim circuit, as shown in Figure 4. Figure 4A shows two circuits capacitively coupled through the stray capacitance between them. In Figure 4B the stray capacitance is intercepted by a grounded Faraday shield, so that interference currents are shunted to ground. For example, a grounded plane can be inserted between PCBs (printed circuit boards) to eliminate most of the capacitive coupling between them.

Another application of the Faraday shield is in the elec-

trostatically shielded transformer. Here, a conducting foil is laid between the primary and secondary coils so as to intercept the capacitive coupling between them. If a system is being upset by AC line transients, this type of transformer may provide the fix. To be effective in this application, the shield must be connected to the green-wire ground.

### SHIELDING AGAINST INDUCTIVE COUPLING

With inductive coupling, the physical mechanism involved is a magnetic flux density  $B$  from some external interference source that links with a current loop in the victim circuit, and generates a voltage in the loop in accordance with Lenz's law:  $v = -NA(dB/dt)$ , where in this case  $N = 1$  and  $A$  is the area of the current loop in the victim circuit.

There are two aspects to defending a circuit against inductive pickup. One aspect is to try to minimize the offensive fields at their source. This is done by minimizing the area of the current loop at the source so as to promote field cancellation, as described in the section on current loops. The other aspect is to minimize the inductive pickup in the victim circuit by minimizing the area of that current loop, since, from Lenz's law, the induced voltage is proportional to this area. So the two aspects really involve the same corrective action: minimize the areas of the current loops. In other words, minimizing the offensiveness of a circuit inherently minimizes its susceptibility.

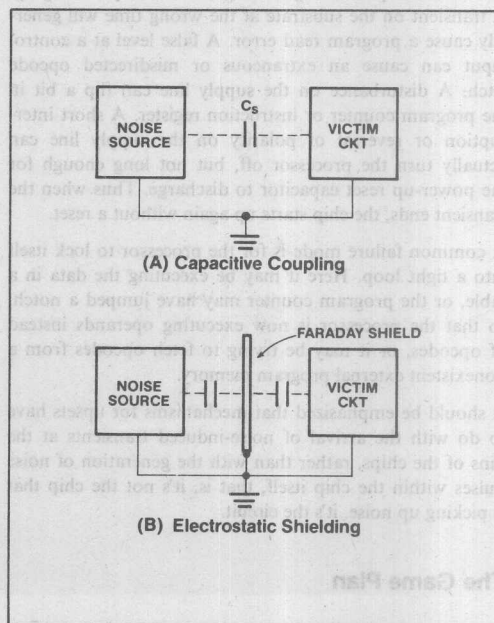


Figure 4. Use of Faraday Shield

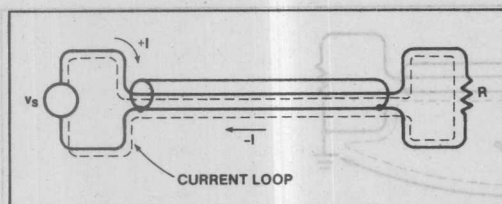


Figure 5. External to the Shield,  $\phi=0$

Shielding against inductive coupling means nothing more nor less than controlling the dimensions of the current loops in the circuit. We must look at four examples of this type of "shielding": the coaxial cable, the twisted pair, the ground plane, and the gridded-ground PCB layout.

**The Coaxial Cable** — Figure 5 shows a coaxial cable carrying a current  $I$  from a signal source to a receiving load. The shield carries the same current as the center conductor. Outside the shield, the magnetic field produced by  $+I$  flowing in the center conductor is cancelled by the field produced by  $-I$  flowing in the shield. To the extent that the cable is ideal in producing zero external magnetic field, it is immune to inductive pickup from external sources. The cable adds effectively zero area to the loop. This is true only if the shield carries the same current as the center conductor.

In the real world, both the signal source and the receiving load are likely to have one end connected to a common signal ground. In that case, should the cable be grounded at one end, both ends, or neither end? The answer is that it should be grounded at both ends. Figure 6A shows the situation when the cable shield is grounded at only one end. In that case the current loop runs down the center conductor of the cable, then back through the common ground connection. The loop area is not well defined. The shield not only does not carry the same current as the center conductor, but it doesn't carry any current at all. There is no field cancellation at all. The shield has no effect whatsoever on either the generation of EMI or susceptibility to EMI. (It is, however, still effective as an electrostatic shield, or at least it would be if the shield coverage were 100%.)

Figure 6B shows the situation when the cable is grounded at both ends. Does the shield carry all of the return current, or only a portion of it on account of the shunting effect of the common ground connection? The answer to that question depends on the frequency content of the signal. In general, the current loop will follow the path of least impedance. At low frequencies, 0Hz to several kHz, where the inductive reactance is insignificant, the current will follow the path of least resistance. Above a few kHz, where inductive reactance predominates, the current will follow the path of least inductance. The path of least

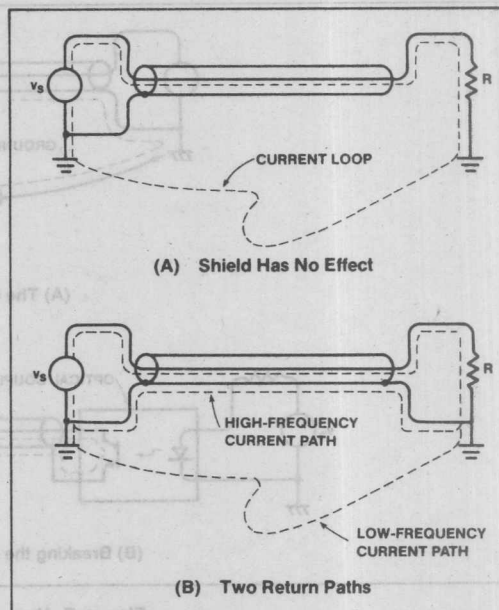


Figure 6. Use of Coaxial Cable

inductance is the path of minimum loop area. Hence, for higher frequencies the shield carries virtually the same current as the center conductor, and is therefore effective against both generation and reception of EMI.

Note that we have now introduced the famous "ground loop" problem, as shown in Figure 7A. Fortunately, a digital system has some built-in immunity to moderate ground loop noise. In a noisy environment, however, one can break the ground loop, and still maintain the shielding effectiveness of the coaxial cable, by inserting an optical coupler, as shown in Figure 7B. What the optical coupler does, basically, is allow us to re-define the signal source as being ungrounded, so that that end of the cable need not be grounded, and still lets the shield carry the same current as the center conductor. Obviously, if the signal source weren't grounded in the first place, the optical coupler wouldn't be needed.

**The Twisted Pair** — A cheaper way to minimize loop area is to run the feed and return wires right next to each other. This isn't as effective as a coaxial cable in minimizing loop area. An ideal coaxial cable adds zero area to the loop, whereas merely keeping the feed and return wires next to each other is bound to add a finite area.

However, two things work to make this cheaper method almost as good as a coaxial cable. First, real coaxial cables are not ideal. If the shield current isn't evenly distributed around the center conductor at every cross-

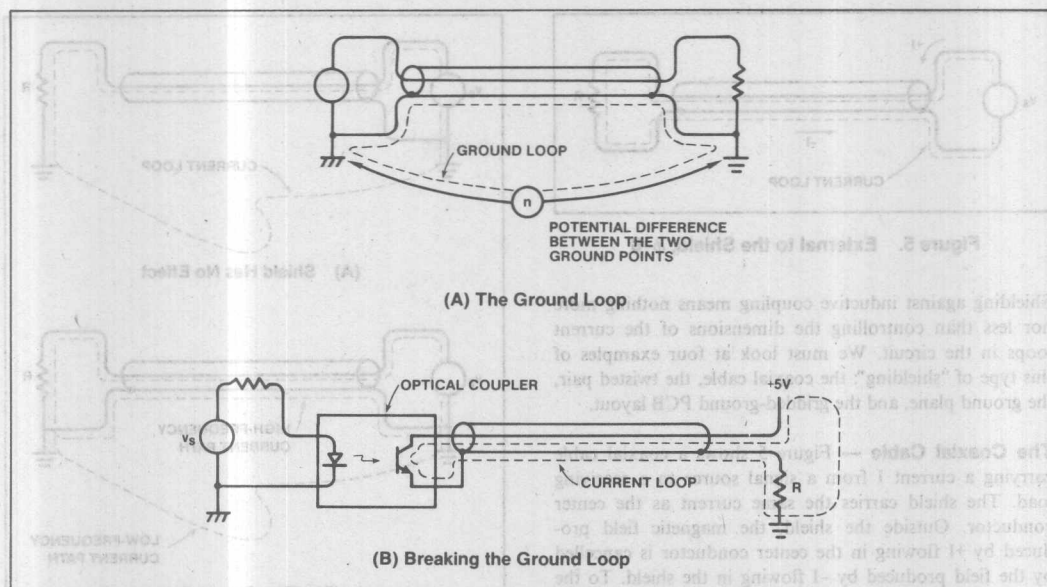


Figure 7. Use of Optical Coupler

section of the cable (it isn't), then field cancellation external to the shield is incomplete. If field cancellation is incomplete, then the effective area added to the loop by the cable isn't zero. Second, in the cheaper method the feed and return wires can be twisted together. This not only maintains their proximity, but the noise picked up in one twist tends to cancel out the noise picked up in the next twist down the line. Thus the "twisted pair" turns out to be about as good a shield against inductive coupling as coaxial cable is.

The twisted pair does not, however, provide electrostatic shielding (i.e., shielding against capacitive coupling). Another operational difference between them is that the coaxial cable works better at higher frequencies. This is primarily because the twisted pair adds more capacitive loading to the signal source than the coaxial cable does. The twisted pair is normally considered useful up to only about 1MHz, as opposed to near a GHz for the coaxial cable.

**The Ground Plane** — The best way to minimize loop areas when many current loops are involved is to use a ground plane. A ground plane is a conducting surface that is to serve as a return conductor for all the current loops in the circuit. Normally, it would be one or more layers of a multilayer PCB. All ground points in the circuit go not to a grounded trace on the PCB, but directly to the ground plane. This leaves each current loop in the circuit free to complete itself in whatever configuration yields minimum loop area (for frequencies wherein the

ground path impedance is primarily inductive).

Thus, if the feed path for a given signal zigzags its way across the PCB, the return path for this signal is free to zigzag right along beneath it on the ground plane, in such a configuration as to minimize the energy stored in the magnetic field produced by this current loop. Minimal magnetic flux means minimal effective loop area and minimal susceptibility to inductive coupling.

**The Gridded-Ground PCB Layout** — The next best thing to a ground plane is to lay out the ground traces on a PCB in the form of a grid structure, as shown in Figure 8. Laying horizontal traces on one side of the board and vertical traces on the other side allows the passage of signal and power traces. Wherever vertical and horizontal ground traces cross, they must be connected by a feed-through.

Have we not created here a network of "ground loops"? Yes, in the literal sense of the word, but loops in the ground layout on a PCB are not to be feared. Such inoffensive little loops have never caused as much noise pick-up as their avoidance has. Trying to avoid innocent little loops in the ground layout, PCB designers have forced current loops into geometries that could swallow a whale. That is exactly the wrong thing to do.

The gridded ground structure works almost as well as the ground plane, as far as minimizing loop area is concerned. For a given current loop, the primary return path may have to zig once in a while where its feed path zags,



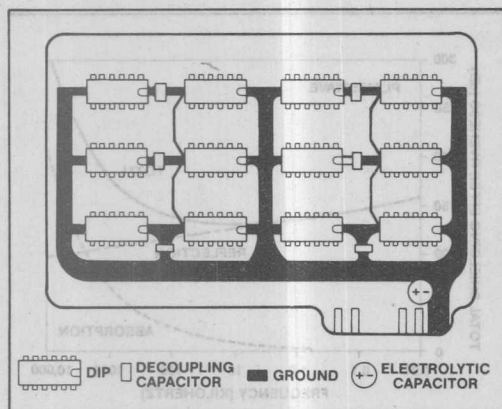


Figure 8. PCB with Gridded Ground

but you still get a mathematically optimal distribution of currents in the grid structure, such that the current loop produces less magnetic flux than if the return path were restrained to follow any single given ground trace. The key to attaining minimum loop areas for all the current loops together is to let the ground currents distribute themselves around the entire area of the board as freely as possible. They want to minimize their own magnetic field. Just let them.

### RF SHIELDING

A time-varying electric field generates a time-varying magnetic field, and vice versa. Far from the source of a time-varying EM field, the ratio of the amplitudes of the electric and magnetic fields is always 377 ohms. Up close to the source of the fields, however, this ratio can be quite different, and dependent on the nature of the source. Where the ratio is near 377 ohms is called the far field, and where the ratio is significantly different from 377 ohms is called the near field. The ratio itself is called the wave impedance,  $E/H$ .

The near field goes out about 1/6 of a wavelength from the source. At 1MHz this is about 150 feet, and at 10MHz it's about 15 feet. That means if an EMI source is in the same room with the victim circuit, it's likely to be a near field problem. The reason this matters is that in the near field an RF interference problem could be almost entirely due to E-field coupling or H-field coupling, and that could influence the choice of an RF shield or whether an RF shield will help at all.

In the near field of a whip antenna, the  $E/H$  ratio is higher than 377 ohms, which means it's mainly an E-field generator. A wire-wrap post can be a whip antenna. Interference from a whip antenna would be by electric field coupling, which is basically capacitive coupling. Methods to protect a circuit from capacitive coupling, such as a Faraday shield, would be effective against RF

interference from a whip antenna. A gridded-ground structure would be less effective.

In the near field of a loop antenna, the  $E/H$  ratio is lower than 377 ohms, which means it's mainly an H-field generator. Any current loop is a loop antenna. Interference from a loop antenna would be by magnetic field coupling, which is basically the same as inductive coupling. Methods to protect a circuit from inductive coupling, such as a gridded-ground structure, would be effective against RF interference from a loop antenna. A Faraday shield would be less effective.

A more difficult case of RF interference, near field or far field, may require a genuine metallic RF shield. The idea behind RF shielding is that time-varying EMI fields induce currents in the shielding material. The induced currents dissipate energy in two ways:  $I^2R$  losses in the shielding material and radiation losses as they re-radiate their own EM fields. The energy for both of these mechanisms is drawn from the impinging EMI fields. Hence the EMI is weakened as it penetrates the shield.

More formally, the  $I^2R$  losses are referred to as absorption loss, and the re-radiation is called reflection loss. As it turns out, absorption loss is the primary shielding mechanism for H-fields, and reflection loss is the primary shielding mechanism for E-fields. Reflection loss, being a surface phenomenon, is pretty much independent of the thickness of the shielding material. Both loss mechanisms, however, are dependent on the frequency ( $\omega$ ) of the impinging EMI field, and on the permeability ( $\mu$ ) and conductivity ( $\sigma$ ) of the shielding material. These loss mechanisms vary approximately as follows:

$$\text{reflection loss to an E-field (in dB)} \sim \log \frac{\sigma}{\omega \mu}$$

$$\text{absorption loss to an H-field (in dB)} \sim t \sqrt{\omega \sigma \mu}$$

where  $t$  is the thickness of the shielding material.

The first expression indicates that E-field shielding is more effective if the shield material is highly conductive, and less effective if the shield is ferromagnetic, and that low-frequency fields are easier to block than high-frequency fields. This is shown in Figure 9.

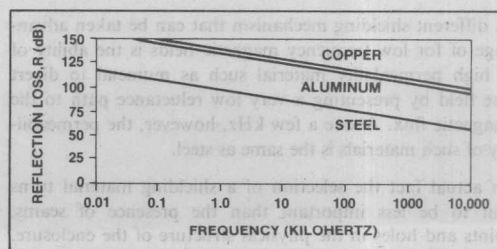
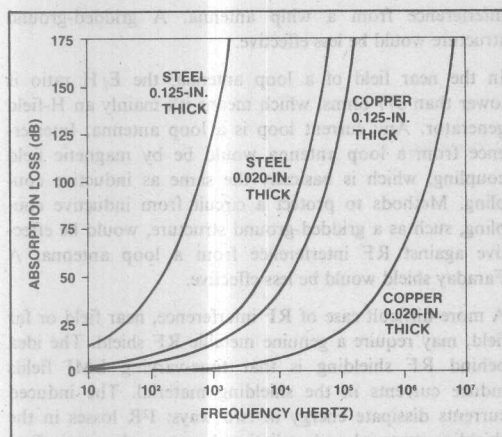


Figure 9. E-Field Shielding





**Figure 10. H-Field Shielding**

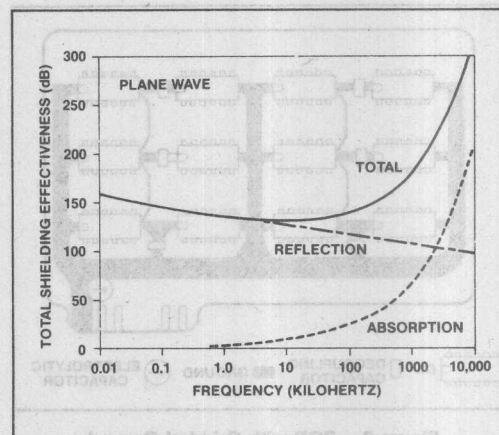
Copper and aluminum both have the same permeability, but copper is slightly more conductive, and so provides slightly greater reflection loss to an E-field. Steel is less effective for two reasons. First, it has a somewhat elevated permeability due to its iron content, and, second, as tends to be the case with magnetic materials, it is less conductive.

On the other hand, according to the expression for absorption loss to an H-field, H-field shielding is more effective at higher frequencies and with shield material that has both high conductivity and high permeability. In practice, however, selecting steel for its high permeability involves some compromise in conductivity. But the increase in permeability more than makes up for the decrease in conductivity, as can be seen in Figure 10. This figure also shows the effect of shield thickness.

A composite of E-field and H-field shielding is shown in Figure 11. However, this type of data is meaningful only in the far field. In the near field the EMI could be 90% H-field, in which case the reflection loss is irrelevant. It would be advisable then to beef up the absorption loss, at the expense of reflection loss, by choosing steel. A better conductor than steel might be less expensive, but quite ineffective.

A different shielding mechanism that can be taken advantage of for low frequency magnetic fields is the ability of a high permeability material such as mumetal to divert the field by presenting a very low reluctance path to the magnetic flux. Above a few kHz, however, the permeability of such materials is the same as steel.

In actual fact the selection of a shielding material turns out to be less important than the presence of seams, joints and holes in the physical structure of the enclosure. The shielding mechanisms are related to the induction of currents in the shield material, but the currents must be



**Figure 11. E- and H-Field Shielding**

allowed to flow freely. If they have to detour around slots and holes, as shown in Figure 12, the shield loses much of its effectiveness.

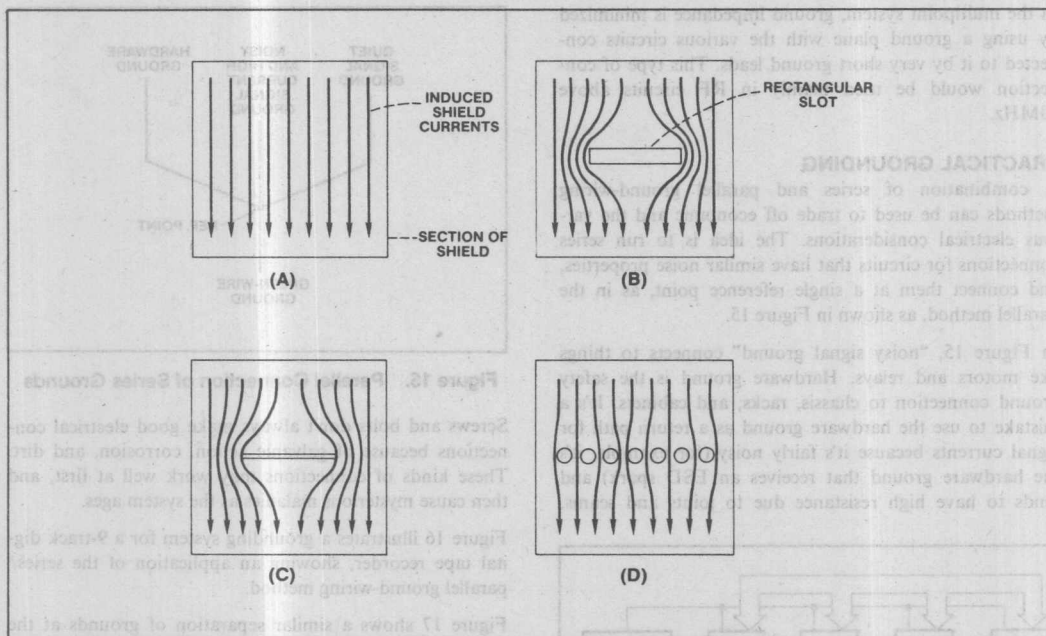
As can be seen in Figure 12, the severity of the detour has less to do with the area of the hole than it does with the geometry of the hole. Comparing Figure 12C with 12D shows that a long narrow discontinuity such as a seam can cause more RF leakage than a line of holes with larger total area. A person who is responsible for designing or selecting rack or chassis enclosures for an EMI environment needs to be familiar with the techniques that are available for maintaining electrical continuity across seams. Information on these techniques is available in the references.

## Grounds

There are two kinds of grounds: earth-ground and signal ground. The earth is not an equipotential surface, so earth ground potential varies. That and its other electrical properties are not conducive to its use as a return conductor in a circuit. However, circuits are often connected to earth ground for protection against shock hazards. The other kind of ground, signal ground, is an arbitrarily selected reference node in a circuit—the node with respect to which other node voltages in the circuit are measured.

## SAFETY GROUND

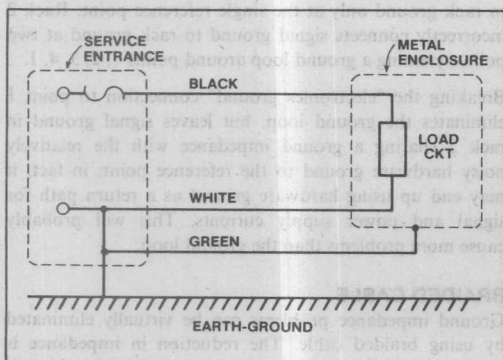
The standard 3-wire single-phase AC power distribution system is represented in Figure 13. The white wire is earth-grounded at the service entrance. If a load circuit has a metal enclosure or chassis, and if the black wire develops a short to the enclosure, there will be a shock hazard to operating personnel, unless the enclosure itself is earth-grounded. If the enclosure is earth-grounded, a



**Figure 12. Effect of Shield Discontinuity on Magnetically Induced Shield Current**

short results in a blown fuse rather than a "hot" enclosure. The earth-ground connection to the enclosure is called a safety ground. The advantage of the 3-wire power system is that it distributes a safety ground along with the power.

Note that the safety-ground wire carries no current, except in case of a fault, so that at least for low frequencies it's at earth-ground potential along its entire length. The white wire, on the other hand, may be several volts off ground, due to the IR drop along its length.



**Figure 13. Single-Phase Power Distribution**

## SIGNAL GROUND

Signal ground is a single point in a circuit that is designated to be the reference node for the circuit. Commonly, wires that connect to this single point are also referred to as "signal ground." In some circles "power supply common" or PSC is the preferred terminology for these conductors. In any case, the manner in which these wires connect to the actual reference point is the basis of distinction among three kinds of signal-ground wiring methods: series, parallel, and multipoint. These methods are shown in Figure 14.

The series connection is pretty common because it's simple and economical. It's the noisiest of the three, however, due to common ground impedance coupling between the circuits. When several circuits share a ground wire, currents from one circuit, flowing through the finite impedance of the common ground line, cause variations in the ground potential of the other circuits. Given that the currents in a digital system tend to be spiked, and that the common impedance is mainly inductive reactance, the variations could be bad enough to cause bit errors in high current or particularly noisy situations.

The parallel connection eliminates common ground impedance problems, but uses a lot of wire. Other disadvantages are that the impedance of the individual ground lines can be very high, and the ground lines themselves can become sources of EM1.

In the multipoint system, ground impedance is minimized by using a ground plane with the various circuits connected to it by very short ground leads. This type of connection would be used mainly in RF circuits above 10MHz.

### PRACTICAL GROUNDING

A combination of series and parallel ground-wiring methods can be used to trade off economic and the various electrical considerations. The idea is to run series connections for circuits that have similar noise properties, and connect them at a single reference point, as in the parallel method, as shown in Figure 15.

In Figure 15, "noisy signal ground" connects to things like motors and relays. Hardware ground is the safety ground connection to chassis, racks, and cabinets. It's a mistake to use the hardware ground as a return path for signal currents because it's fairly noisy (for example, it's the hardware ground that receives an ESD spark) and tends to have high resistance due to joints and seams.

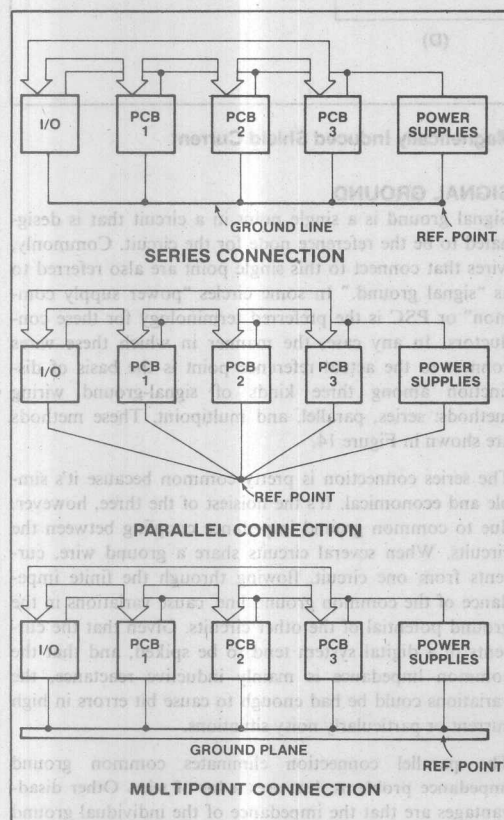


Figure 14. Three Ways to Wire the Grounds

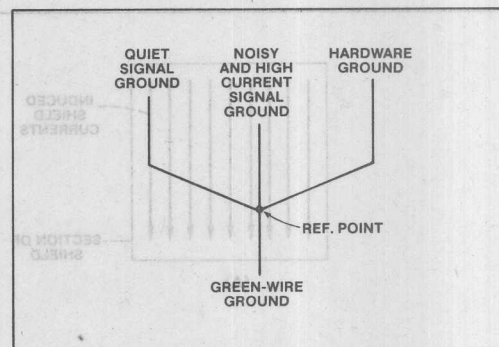


Figure 15. Parallel Connection of Series Grounds

Screws and bolts don't always make good electrical connections because of galvanic action, corrosion, and dirt. These kinds of connections may work well at first, and then cause mysterious maladies as the system ages.

Figure 16 illustrates a grounding system for a 9-track digital tape recorder, showing an application of the series/parallel ground-wiring method.

Figure 17 shows a similar separation of grounds at the PCB level. Currents in multiplexed LED displays tend to put a lot of noise on the ground and supply lines because of the constant switching and changing involved in the scanning process. The segment driver ground is relatively quiet, since it doesn't conduct the LED currents. The digit driver ground is noisier, and should be provided with a separate path to the PCB ground terminal, even if the PCB ground layout is gridded. The LED feed and return current paths should be laid out on opposite sides of the board like parallel flat conductors.

Figure 18 shows right and wrong ways to make ground connections in racks. Note that the safety ground connections from panel to rack are made through ground straps, not panel screws. Rack 1 correctly connects signal ground to rack ground only at the single reference point. Rack 2 incorrectly connects signal ground to rack ground at two points, creating a ground loop around points 1, 2, 3, 4, 1.

Breaking the "electronics ground" connection to point 1 eliminates the ground loop, but leaves signal ground in rack 2 sharing a ground impedance with the relatively noisy hardware ground to the reference point; in fact, it may end up using hardware ground as a return path for signal and power supply currents. This will probably cause more problems than the ground loop.

### BRAIDED CABLE

Ground impedance problems can be virtually eliminated by using braided cable. The reduction in impedance is due to skin effect: At higher frequencies the current tends to flow along the surface of a conductor rather than uni-

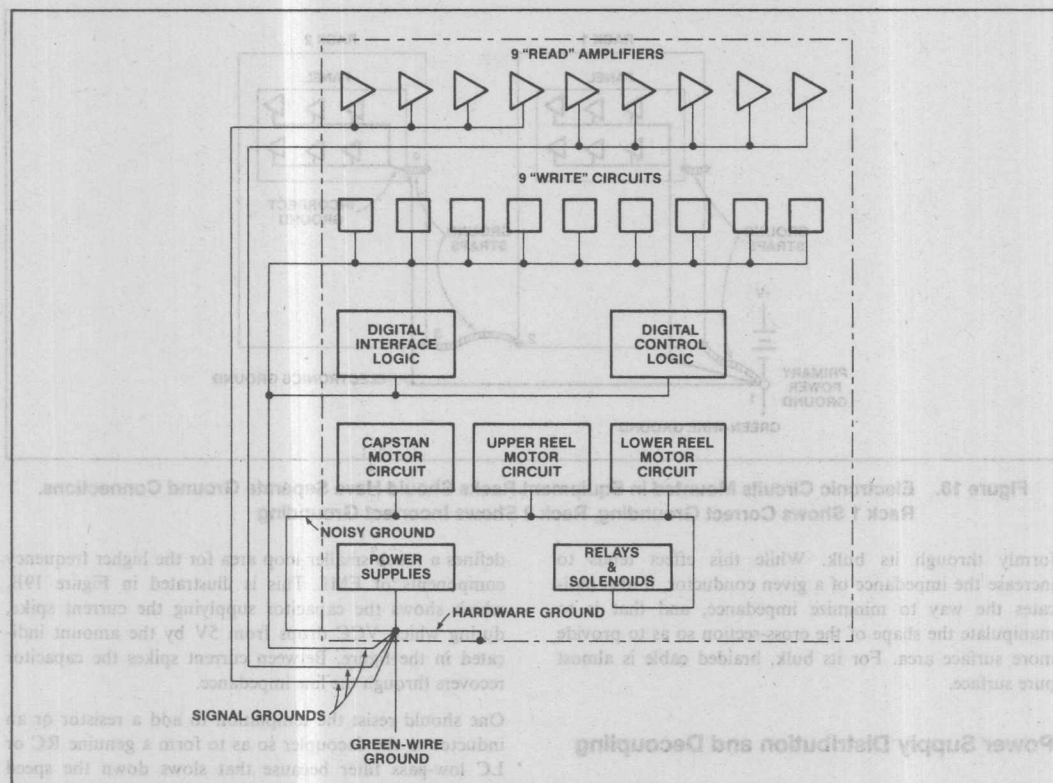


Figure 16. Ground System in a 9-Track Digital Recorder

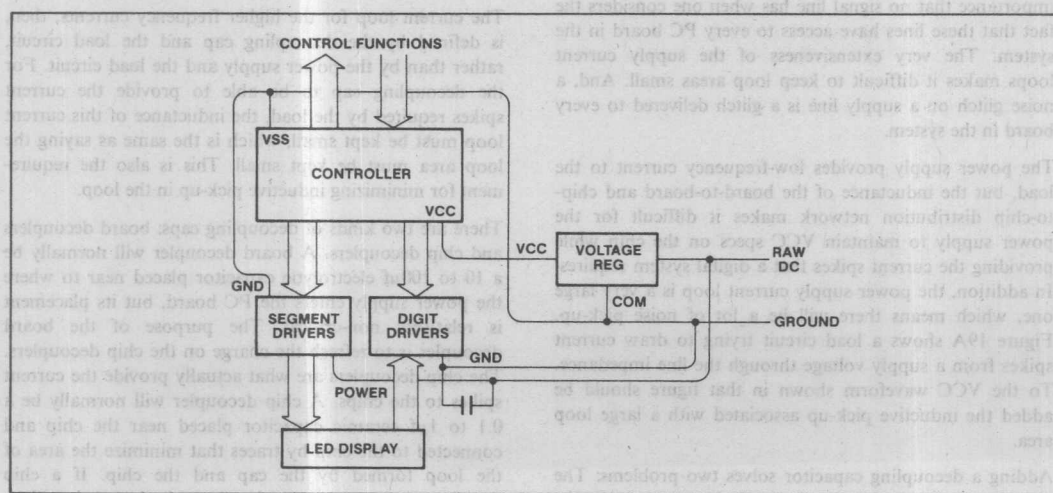
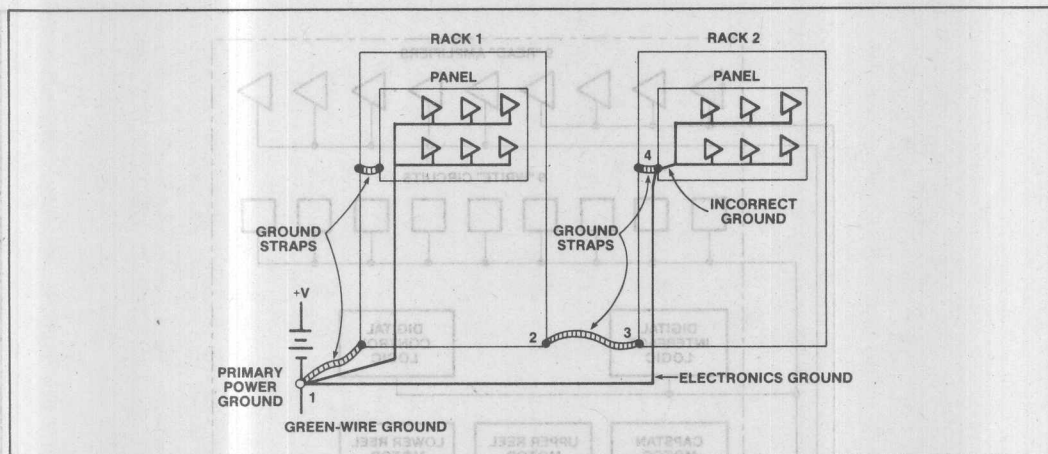


Figure 17. Separate Ground for Multiplexed LED Display





**Figure 18. Electronic Circuits Mounted in Equipment Racks Should Have Separate Ground Connections. Rack 1 Shows Correct Grounding, Rack 2 Shows Incorrect Grounding**

formly through its bulk. While this effect tends to increase the impedance of a given conductor, it also indicates the way to minimize impedance, and that is to manipulate the shape of the cross-section so as to provide more surface area. For its bulk, braided cable is almost pure surface.

### Power Supply Distribution and Decoupling

The main consideration for power supply distribution lines is, as for signal lines, to minimize the areas of the current loops. But the power supply lines take on an importance that no signal line has when one considers the fact that these lines have access to every PC board in the system. The very extensiveness of the supply current loops makes it difficult to keep loop areas small. And, a noise glitch on a supply line is a glitch delivered to every board in the system.

The power supply provides low-frequency current to the load, but the inductance of the board-to-board and chip-to-chip distribution network makes it difficult for the power supply to maintain VCC specs on the chip while providing the current spikes that a digital system requires. In addition, the power supply current loop is a very large one, which means there will be a lot of noise pick-up. Figure 19A shows a load circuit trying to draw current spikes from a supply voltage through the line impedance. To the VCC waveform shown in that figure should be added the inductive pick-up associated with a large loop area.

Adding a decoupling capacitor solves two problems: The capacitor acts as a nearby source of charge to supply the current spikes through a smaller line impedance, and it

defines a much smaller loop area for the higher frequency components of EMI. This is illustrated in Figure 19B, which shows the capacitor supplying the current spike, during which VCC drops from 5V by the amount indicated in the figure. Between current spikes the capacitor recovers through the line impedance.

One should resist the temptation to add a resistor or an inductor to the decoupler so as to form a genuine RC or LC low-pass filter because that slows down the speed with which the decoupler cap can be refreshed. Good filtering and good decoupling are not necessarily the same thing.

The current loop for the higher frequency currents, then, is defined by the decoupling cap and the load circuit, rather than by the power supply and the load circuit. For the decoupling cap to be able to provide the current spikes required by the load, the inductance of this current loop must be kept small, which is the same as saying the loop area must be kept small. This is also the requirement for minimizing inductive pick-up in the loop.

There are two kinds of decoupling caps: board decouplers and chip decouplers. A board decoupler will normally be a 10 to 100 $\mu$ F electrolytic capacitor placed near to where the power supply enters the PC board, but its placement is relatively non-critical. The purpose of the board decoupler is to refresh the charge on the chip decouplers. The chip decouplers are what actually provide the current spikes to the chips. A chip decoupler will normally be a 0.1 to 1 $\mu$ F ceramic capacitor placed near the chip and connected to the chip by traces that minimize the area of the loop formed by the cap and the chip. If a chip decoupler is not properly placed on the board, it will be ineffective as a decoupler and will serve only to increase

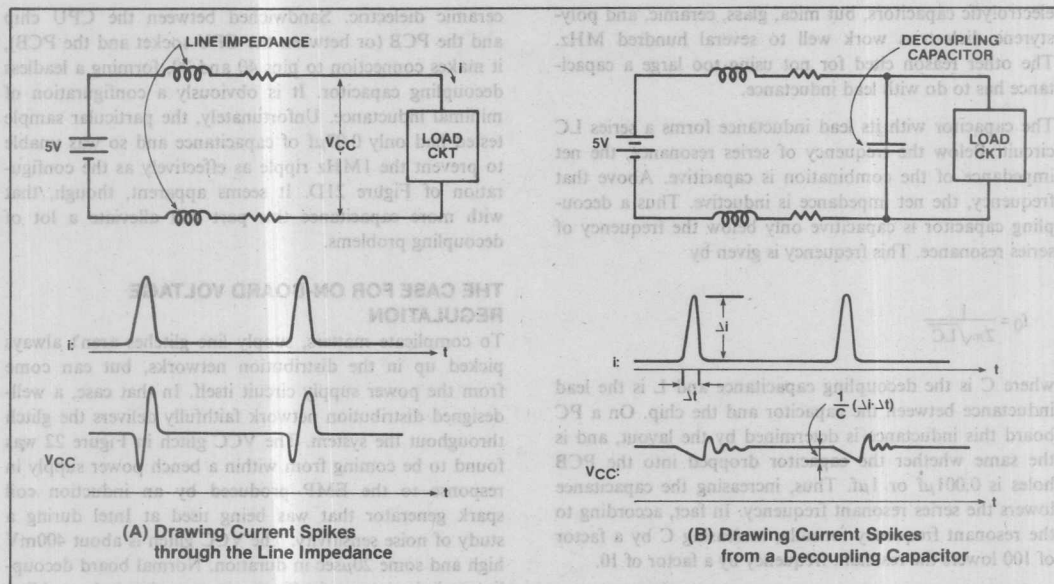


Figure 19. What a Decoupling Capacitor Does

the cost of the board. Good and bad placement of decoupling capacitors are illustrated in Figure 20.

Power distribution traces on the PC board need to be laid out so as to obtain minimal area (minimal inductance) in the loops formed by each chip and its decoupler, and by the chip decouplers and the board decoupler. One way to accomplish this goal is to use a power plane. A power plane is the same as a ground plane, but at VCC potential. More economically, a power grid similar to the ground grid previously discussed (Figure 8) can be used. Actually, if the chip decoupling loops are small, other aspects of the power layout are less critical. In other words, power planes and power gridding aren't needed, but power traces *should* be laid in the closest possible proximity to ground traces, preferably so that

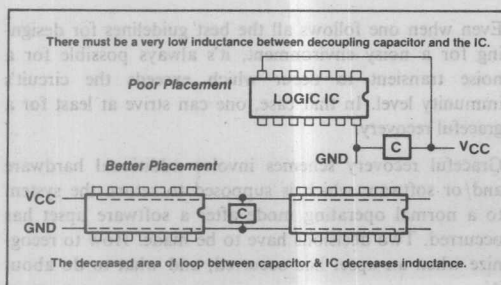


Figure 20. Placement of Decoupling Capacitors

each power trace is on the direct opposite side of the board from a ground trace.

Special-purpose power supply distribution buses which mount on the PCB are available. The buses use a parallel flat conductor configuration, one conductor being a VCC line and the other a ground line. Used in conjunction with a gridded ground layout, they not only provide a low-inductance distribution system, but can themselves form part of the ground grid, thus facilitating the PCB layout. The buses are available with and without enhanced bus capacitance, under the names Mini/Bus® and Q/PAC® from Rogers Corp. (5750 E. McKellips, Mesa, AZ 85205).

## SELECTING THE VALUE OF THE DECOUPLING CAP

The effectiveness of the decoupling cap has a lot to do with the way the power and ground traces connect this capacitor to the chip. In fact, the area formed by this loop is more important than the value of the capacitance. Then, given that the area of this loop is indeed minimal, it can generally be said that the larger the value of the decoupling cap, the more effective it is, if the cap has a mica, ceramic, glass, or polystyrene dielectric.

It's often said, and not altogether accurately, that the chip decoupler shouldn't have too large a value. There are two reasons for this statement. One is that some capacitors, because of the nature of their dielectrics, tend to become inductive or lossy at higher frequencies. This is true of

electrolytic capacitors, but mica, glass, ceramic, and polystyrene dielectrics work well to several hundred MHz. The other reason cited for not using too large a capacitance has to do with lead inductance.

The capacitor with its lead inductance forms a series LC circuit. Below the frequency of series resonance, the net impedance of the combination is capacitive. Above that frequency, the net impedance is inductive. Thus a decoupling capacitor is capacitive only below the frequency of series resonance. This frequency is given by

$$f_0 = \frac{1}{2\pi\sqrt{LC}}$$

where C is the decoupling capacitance and L is the lead inductance between the capacitor and the chip. On a PC board this inductance is determined by the layout, and is the same whether the capacitor dropped into the PCB holes is 0.001  $\mu$ f or 1  $\mu$ f. Thus, increasing the capacitance lowers the series resonant frequency. In fact, according to the resonant frequency formula, increasing C by a factor of 100 lowers the resonant frequency by a factor of 10.

Figures quoted on the series resonant frequency of a 0.01  $\mu$ f capacitor run from 10 to 15 MHz, depending on the lead length. If these numbers were accurate, a 1  $\mu$ f capacitor in the same position on the board would have a resonant frequency of 1.0 to 1.5 MHz, and as a decoupler would do more harm than good. However, the numbers are based on a presumed inductance of a given length of wire (the lead length). It should be noted that a "length of wire" has no inductance at all, strictly speaking. Only a complete current loop has inductance, and the inductance depends on the geometry of the loop. Figures quoted on the inductance of a length of wire are based on a presumably "very large" loop area, such that the magnetic field produced by the return current has no cancellation effect on the field produced by the current in the given length of wire. Such a loop geometry is not and should not be the case with the decoupling loop.

Figure 21 shows VCC waveforms, measured between pins 40 and 20 (VCC and VSS) of an 8751 CPU, for several conditions of decoupling on a PC board that has a decoupling loop area slightly larger than necessary. These photographs show the effects of increasing the decoupling capacitance and decreasing the area of the decoupling loop. The indications are that a 1  $\mu$ f capacitor is better than a 0.1  $\mu$ f capacitor, which in turn is better than nothing, and that the board should have been laid out with more attention paid to the area of the decoupling loop.

Figure 21E was obtained using a special-purpose experimental capacitor designed by Rogers Corp. (Q-Pac Division, Mesa, AZ) for use as a decoupler. It consists of two parallel plates, the length of a 40-pin DIP, separated by a

ceramic dielectric. Sandwiched between the CPU chip and the PCB (or between the CPU socket and the PCB), it makes connection to pins 40 and 20, forming a leadless decoupling capacitor. It is obviously a configuration of minimal inductance. Unfortunately, the particular sample tested had only 0.07  $\mu$ f of capacitance and so was unable to prevent the 1 MHz ripple as effectively as the configuration of Figure 21D. It seems apparent, though, that with more capacitance this part will alleviate a lot of decoupling problems.

## THE CASE FOR ON-BOARD VOLTAGE REGULATION

To complicate matters, supply line glitches aren't always picked up in the distribution networks, but can come from the power supply circuit itself. In that case, a well-designed distribution network faithfully delivers the glitch throughout the system. The VCC glitch in Figure 22 was found to be coming from within a bench power supply in response to the EMP produced by an induction coil spark generator that was being used at Intel during a study of noise sensitivity. The VCC glitch is about 400 mV high and some 20  $\mu$ sec in duration. Normal board decoupling techniques were ineffective in removing it, but adding an on-board voltage regulator chip did the job.

Thus, a good case can be made in favor of using a voltage regulator chip on each PCB, instead of doing all the voltage regulation at the supply circuit. This eases requirements on the heat-sinking at the supply circuit, and alleviates much of the distribution and board decoupling headaches. However, it also brings in the possibility that different boards would be operating at slightly different VCC levels due to tolerance in the regulator chips; this then leads to slightly different logic levels from board to board. The implications of that may vary from nothing to latch-up, depending on what kinds of chips are on the boards, and how they react to an input "high" that is perhaps 0.4 V higher than local VCC.

## Recovering Gracefully from a Software Upset

Even when one follows all the best guidelines for designing for a noisy environment, it's always possible for a noise transient to occur which exceeds the circuit's immunity level. In that case, one can strive at least for a graceful recovery.

Graceful recovery schemes involve additional hardware and/or software which is supposed to return the system to a normal operating mode after a software upset has occurred. Two decisions have to be made: How to recognize when an upset has occurred, and what to do about it.

If the designer knows what kinds and combinations of

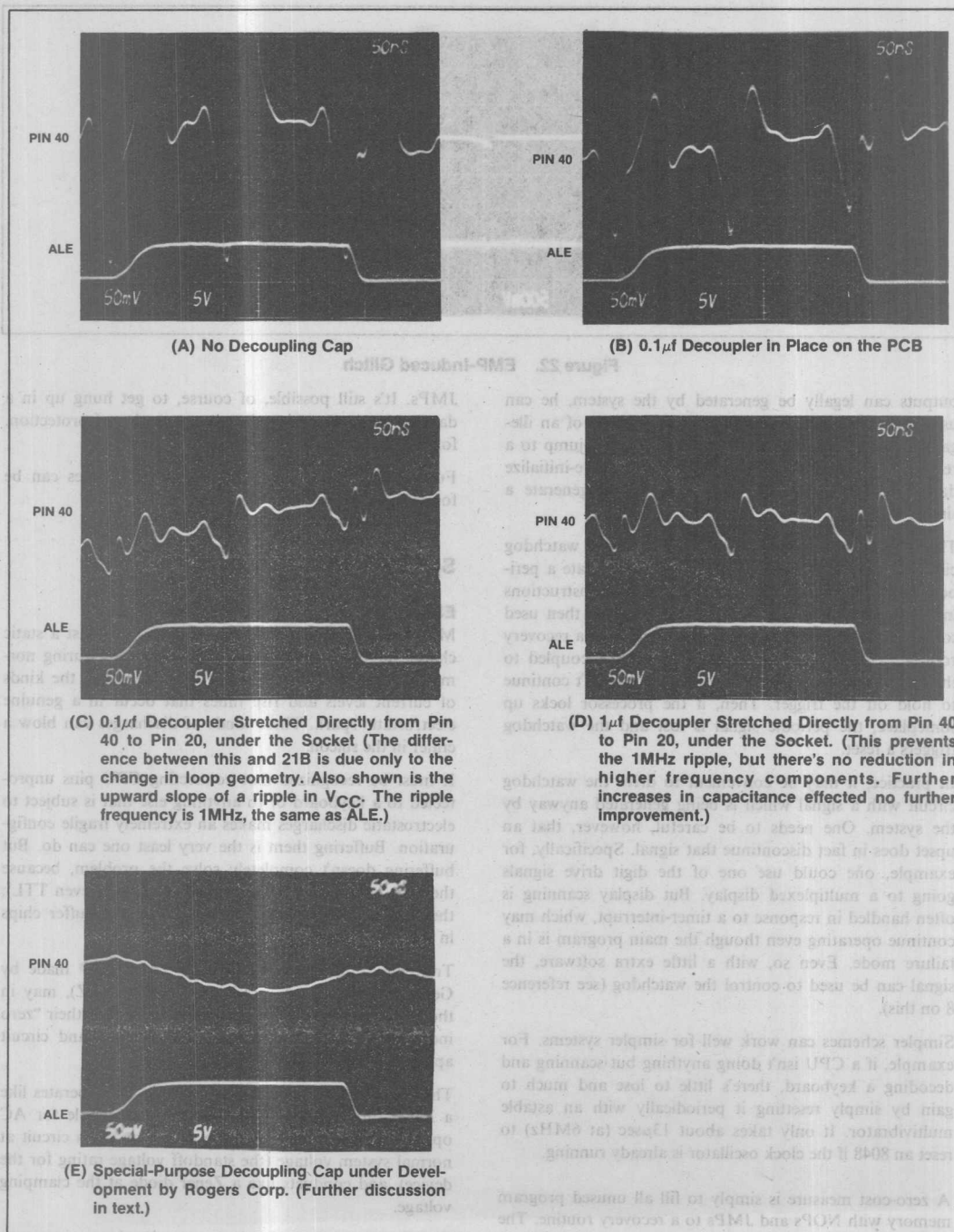


Figure 21. Noise on  $V_{CC}$  Line



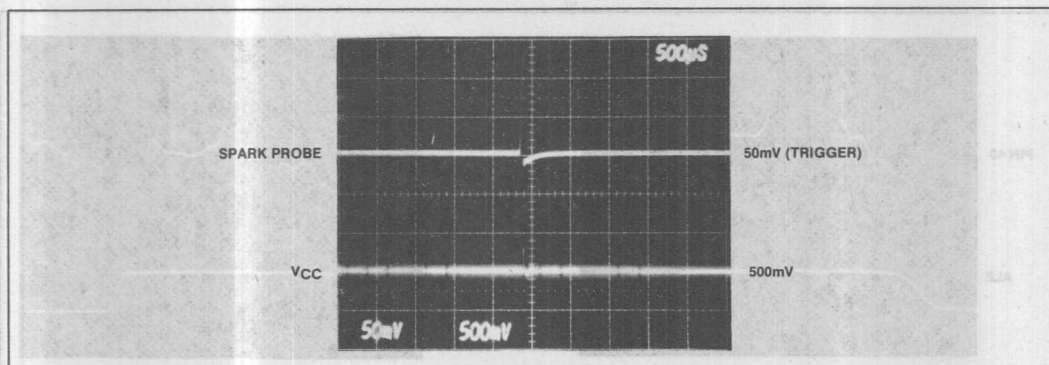


Figure 22. EMP-Induced Glitch

outputs can legally be generated by the system, he can use gates to recognize and flag the occurrence of an illegal state of affairs. The flag can then trigger a jump to a recovery routine which then may check or re-initialize data, perhaps output an error message, or generate a simple reset.

The most reliable scheme is to use a so-called watchdog circuit. Here the CPU is programmed to generate a periodic signal as long as the system is executing instructions in an expected manner. The periodic signal is then used to hold off a circuit that will trigger a jump to a recovery routine. The periodic signal needs to be AC-coupled to the trigger circuit so that a "stuck-at" fault won't continue to hold off the trigger. Then, if the processor locks up someplace, the periodic signal is lost and the watchdog triggers a reset.

In practice, it may be convenient to drive the watchdog circuit with a signal which is being generated anyway by the system. One needs to be careful, however, that an upset does in fact discontinue that signal. Specifically, for example, one could use one of the digit drive signals going to a multiplexed display. But display scanning is often handled in response to a timer-interrupt, which may continue operating even though the main program is in a failure mode. Even so, with a little extra software, the signal can be used to control the watchdog (see reference 8 on this).

Simpler schemes can work well for simpler systems. For example, if a CPU isn't doing anything but scanning and decoding a keyboard, there's little to lose and much to gain by simply resetting it periodically with an astable multivibrator. It only takes about  $13\mu\text{sec}$  (at 6MHz) to reset an 8048 if the clock oscillator is already running.

A zero-cost measure is simply to fill all unused program memory with NOPs and JMPs to a recovery routine. The effectiveness of this method is increased by writing the program in segments that are separated by NOPs and

JMPs. It's still possible, of course, to get hung up in a data table or something. But you get a lot of protection, for the cost.

Further discussion of graceful recovery schemes can be found in reference 13.

## Special Problem Areas

### ESD

MOS chips have some built-in protection against a static charge build-up on the pins, as would occur during normal handling, but there's no protection against the kinds of current levels and rise times that occur in a genuine electrostatic spark. These kinds of discharges can blow a crater in the silicon.

It must be recognized that connecting CPU pins unprotected to a keyboard or to anything else that is subject to electrostatic discharges makes an extremely fragile configuration. Buffering them is the very least one can do. But buffering doesn't completely solve the problem, because then the buffer chips will sustain the damage (even TTL); therefore, one might consider mounting the buffer chips in sockets for ease of replacement.

Transient suppressors, such as the TranZorbs® made by General Semiconductor Industries (Tempe, AZ), may in the long run provide the cheapest protection if their "zero inductance" structure is used. The structure and circuit application are shown in Figure 23.

The suppressor element is a pn junction that operates like a Zener diode. Back-to-back units are available for AC operation. The element is more or less an open circuit at normal system voltage (the standoff voltage rating for the device), and conducts like a Zener diode at the clamping voltage.

The lead inductance in the conventional transient suppressor package makes the conventional package essen-

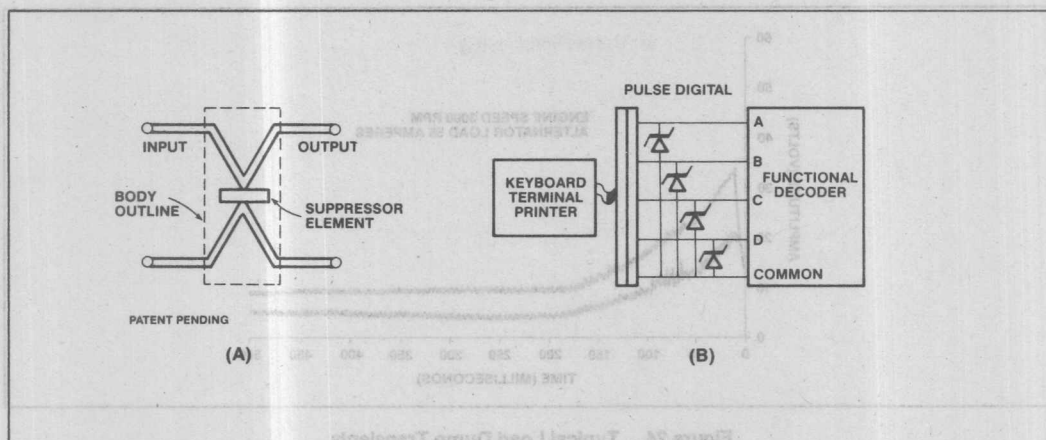


Figure 23. "Zero-inductance" Structure and Use in Circuit

tially useless for protection against ESD pulses, owing to the fast rise of these pulses. The "zero inductance" units are available singly in a 4-pin DIP, and in arrays of four to a 16-pin DIP for PCB level protection. In that application they should be mounted in close proximity to the chips they protect.

In addition, metal enclosures or frames or parts that can receive an ESD spark should be connected by braided cable to the green-wire ground. Because of the ground impedance, ESD current shouldn't be allowed to flow through any signal ground, even if the chips are protected by transient suppressors. A 35kV ESD spark can always spare a few hundred volts to drive a fast current pulse down a signal ground line if it can't find a braided cable to follow. Think how delighted your 8048 will be to find its VSS pin about 250V higher than VCC for a few 10s of nanoseconds.

### THE AUTOMOTIVE ENVIRONMENT

The automobile presents an extremely hostile environment for electronic systems. There are several parts to it:

1. Temperature extremes from  $-40^{\circ}\text{C}$  to  $+125^{\circ}\text{C}$  (under the hood) or  $+85^{\circ}\text{C}$  (in the passenger compartment)
2. Electromagnetic pulses from the ignition system
3. Supply line transients that will knock your socks off

One needs to take a long, careful look at the temperature extremes. The allowable storage temperature range for most Intel MOS chips is  $-65^{\circ}\text{C}$  to  $+150^{\circ}\text{C}$ , although some chips have a maximum storage temperature rating of  $+125^{\circ}\text{C}$ . In operation (or "under bias," as the data sheets say) the allowable ambient temperature range depends on the product grade, as follows:

Grade	Ambient Temperature	
	min.	max.
Commercial	0	70
Industrial	$-40$	$+85$
Automotive	$-40$	$+110$
Military	$-55$	$+125$

The different product grades are actually the same chip, but tested according to different standards. Thus, a given commercial-grade chip might actually pass military temperature requirements, but not have been tested for it. (Of course, there are other differences in grading requirements having to do with packaging, burn-in, traceability, etc.)

In any case, it's apparent that commercial-grade chips can't be used safely in automotive applications, not even in the passenger compartment. Industrial-grade chips can be used in the passenger compartment, and automotive or military chips are required in under-the-hood applications.

Ignition noise, CB radios, and that sort of thing are probably the least of your worries. In a poorly designed system, or in one that has not been adequately tested for the automotive environment, this type of EMI might cause a few software upsets, but not destroy chips.

The major problem, and the one that seems to come as the biggest surprise to most people, is the line transients. Regrettably, the 12V battery is not actually the source of power when the car is running. The charging system is, and it's not very clean. The only time the battery is the real source of power is when the car is first being started, and in that condition the battery terminals may be delivering about 5 or 6V. Below is a brief description of the major idiosyncracies of the "12V" automotive power line.

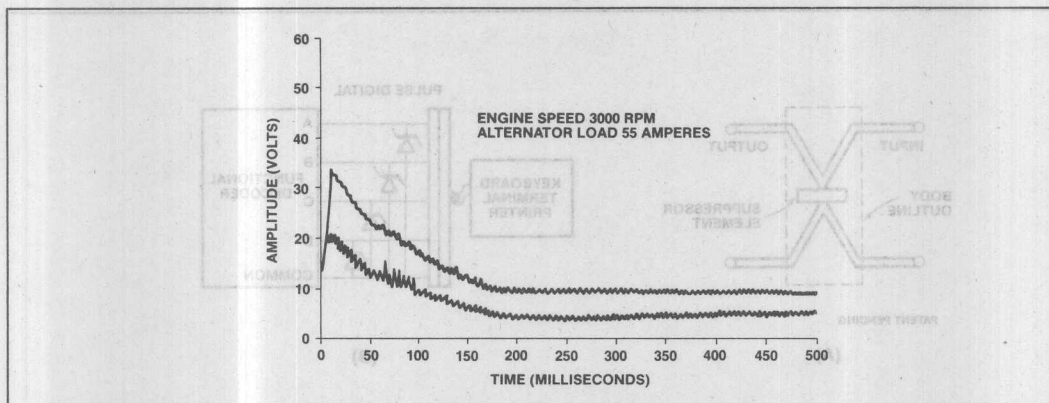


Figure 24. Typical Load Dump Transients

- An abrupt reduction in the alternator load causes a positive voltage transient called "load dump." In a load dump transient the line voltage rises to 20 or 30V in a few msec, then decays exponentially with a time constant of about 100msec, as shown in Figure 24. Much higher peak voltages and longer decay times have also been reported. The worst case load dump is caused by disconnecting a low battery from the alternator circuit while the alternator is running. Normally this would happen intermittently when the battery terminal connections are defective.
- When the ignition is turned off, as the field excitation decays, the line voltage can go to between -40 and -100V for 100 msec or more.
- Miscellaneous solenoid switching transients, such as the one shown in Figure 25, can drive the line to + or -200 to 400V for several  $\mu$ sec.
- Mutual coupling between unshielded wires in long harnesses can induce 100 and 200V transients in unprotected circuits.

What all this adds up to is that people in the business of building systems for automotive applications need a comprehensive testing program. An SAE guideline which describes the automotive environment is available to designers: SAE J1211, "Recommended Environmental Practices for Electronic Equipment Design," 1980 SAE Handbook, Part 1, pp. 22.80-22.96.

Some suggestions for protecting circuitry are shown in Figure 26. A transient suppressor is placed in front of the regulator chip to protect it. Since the rise times in these transients are not like those in ESD pulses, lead inductance is less critical and conventional devices can be used. The regulator itself is pretty much of a necessity, since a load dump transient is simply not going to be removed

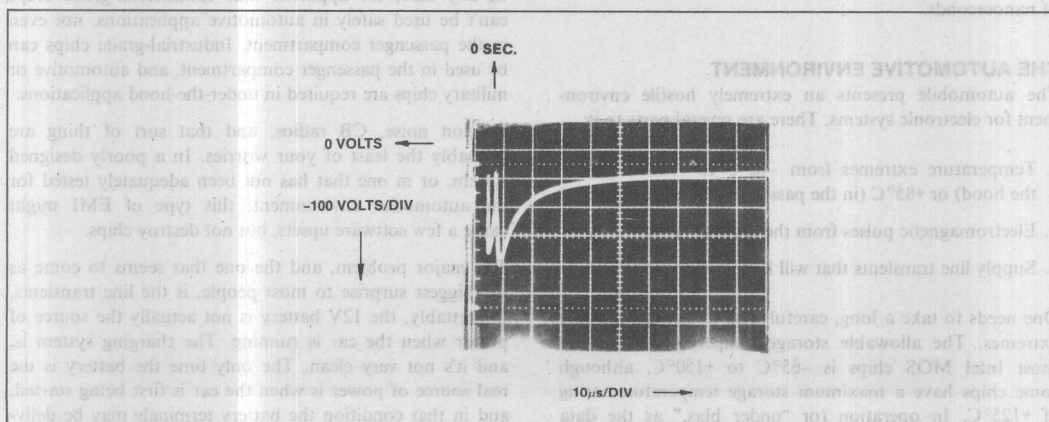


Figure 25. Transient Created by De-energizing an Air Conditioning Clutch Solenoid

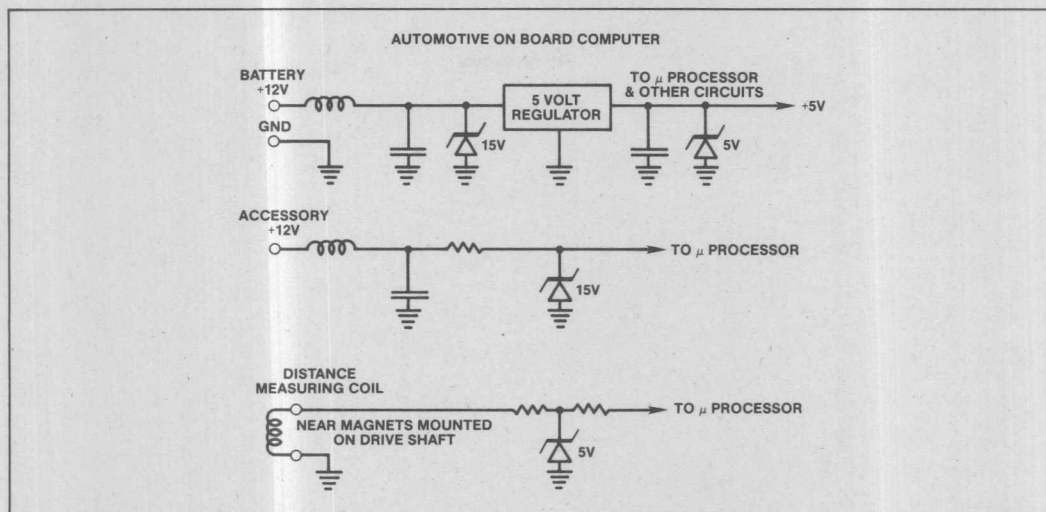


Figure 26. Use of Transient Suppressors in Automotive Applications

by any conventional LC or RC filter.

Special I/O interfacing is also required, because of the need for high tolerance to voltage transients, input noise, input/output isolation, etc. In addition, switches that are being monitored or driven by these buffers are usually referenced to chassis ground instead of signal ground, and in a car there can be many volts difference between the two. I/O interfacing is discussed in reference 2.

The EMC Education committee has available a video tape: "Introduction to EMC — A Video Training Tape," by Henry Ott. Don White Consultants offers a series of training courses on many different aspects of electromagnetic compatibility. Most organizations that sponsor EMC courses also offer in-plant presentations.

## Parting Thoughts

The main sources of information for this Application Note were the references by Ott and by White. Reference 5 is probably the finest treatment currently available on the subject. The other references provided specific information as cited in the text.

Courses and seminars on the subject of electromagnetic interference are given regularly throughout the year. Information on these can be obtained from:

IEEE Electromagnetic Compatibility Society  
EMC Education Committee  
345 East 47th Street  
New York, NY 10017  
Phone: (212) 752-6800

Don White Consultants, Inc.  
International Training Centre  
P.O. Box D  
Gainesville, VA 22065  
Phone: (703) 347-0030



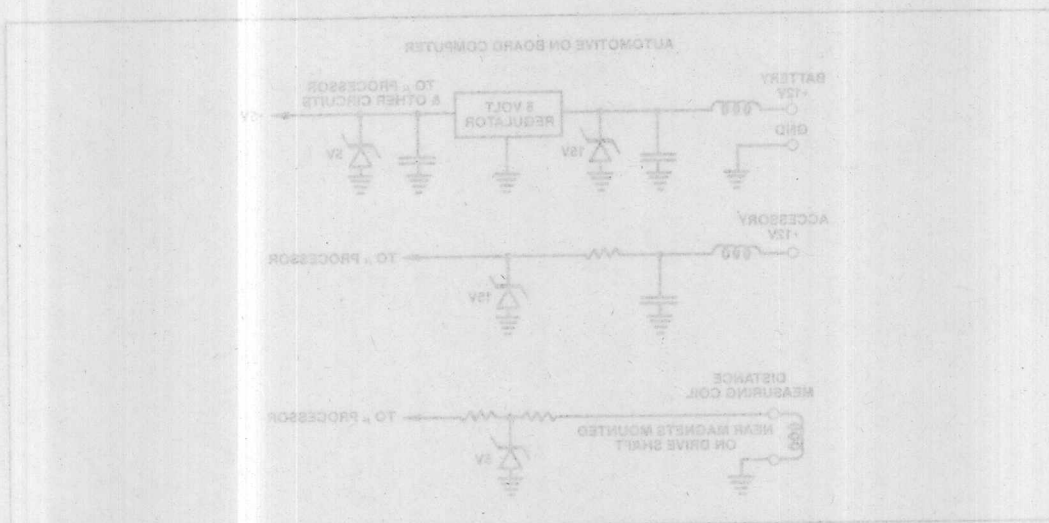


Figure 26. Use of Transient Suppressors in Automotive Applications

The EMC Education Committee has available a video tape "Introduction to EMC — A Video Training Tape" by Henry Ott. Don White Consultants offers a series of training courses on many different aspects of electromagnetic compatibility. Most organizations that sponsor EMC courses also offer in-plant presentations.

Special I/O interfacing is also required because of the need for high tolerance to voltage transients, input noise, input/output isolation, etc. In addition, switches that are monitored or driven by these buffers are usually referenced to chassis ground instead of signal ground, and in a car there can be many volts difference between the two. I/O interfacing is discussed in reference 2.

## Fairing Thoughts

The main source of information for this Application Note were the references by Ott and by White. Reference 2 is probably the finest treatment currently available on the subject. The other references provided specific information as cited in the text.

Courses and seminars on the subject of electromagnetic interference are given regularly throughout the year. Information on these can be obtained from:

IEEE Electromagnetic Compatibility Society  
EMC Education Committee  
445 East 47th Street  
New York, NY 10017  
Phone: (212) 732-6800

Don White Consultants, Inc.  
International Training Centre  
P.O. Box D  
Gainesville, VA 22063  
Phone: (703) 347-0030



# APPLICATION NOTE

AP-155

June 1983

## Oscillators for Microcontrollers

Tom Williamson  
Microcontroller  
Technical Marketing

Order Number: 230659-001

# **Oscillators**

## **for Microcontrollers**

## **CONTENTS**

<b>INTRODUCTION</b> .....	22-25
---------------------------	-------

### **FEEDBACK OSCILLATORS**

Loop Gain .....	22-25
How Feedback Oscillators Work .....	22-26
The Positive Reactance Oscillator ...	22-26

### **QUARTZ CRYSTALS**

Crystal Parameters .....	22-27
equivalent circuit .....	22-27
load capacitance .....	22-27
"series" vs. "parallel" crystals ..	22-28
equivalent series resistance .....	22-28
frequency tolerance .....	22-28
drive level .....	22-29

### **CERAMIC RESONATORS** .....

Specifications for Ceramic Resonators .....	22-30
---	-------

### **OSCILLATOR DESIGN CONSIDERATIONS**

On-Chip Oscillators .....	22-30
crystal specifications .....	22-30
oscillation frequency .....	22-30
selection of CX1 and CX2 .....	22-31
placement of components .....	22-31
clocking other chips .....	22-31
External Oscillators .....	22-31
gate oscillators vs. discrete devices .....	22-33
fundamental vs. overtone operation .....	22-33
"series" vs. "parallel" operation .....	22-33

### **MORE ABOUT USING THE "ON-CHIP" OSCILLATORS**

Oscillator Calculations .....	22-34
Start-Up Characteristics .....	22-35
Steady-State Characteristics .....	22-37
Pin Capacitance .....	22-38
MCS®-51 Oscillator .....	22-39
MCS®-48 Oscillator .....	22-39
Pre-Production Tests .....	22-42
troubleshooting oscillator problems .....	22-43

### **APPENDIX I**

Quartz and Ceramic Resonator Formulas .....	22-46
---	-------

### **APPENDIX II**

Oscillator Analysis Program .....	22-48
-----------------------------------	-------

## INTRODUCTION

Intel's microcontroller families (MCS®-48, MCS-51, and iACX-96) contain a circuit that is commonly referred to as the "on-chip oscillator". The on-chip circuitry is not itself an oscillator, of course, but an amplifier that is suitable for use as the amplifier part of a feedback oscillator. The data sheets and Microcontroller Handbook show how the on-chip amplifier and several off-chip components can be used to design a working oscillator. With proper selection of off-chip components, these oscillator circuits will perform better than almost any other type of clock oscillator, and by almost any criterion of excellence. The suggested circuits are simple, economical, stable, and reliable.

We offer assistance to our customers in selecting suitable off-chip components to work with the on-chip oscillator circuitry. It should be noted, however, that Intel cannot assume the responsibility of writing specifications for the off-chip components of the complete oscillator circuit, nor of guaranteeing the performance of the finished design in production, anymore than a transistor manufacturer, whose data sheets show a number of suggested amplifier circuits, can assume responsibility for the operation, in production, of any of them.

We are often asked why we don't publish a list of required crystal or ceramic resonator specifications, and recommend values for the other off-chip components. This has been done in the past, but sometimes with consequences that were not intended.

Suppose we suggest a maximum crystal resistance of 30 ohms for some given frequency. Then your crystal supplier tells you the 30-ohm crystals are going to cost twice as much as 50-ohm crystals. Fearing that Intel will not "guarantee operation" with 50-ohm crystals, you order the expensive ones. In fact, Intel guarantees only what is embodied within an Intel product. Besides, there is no reason why 50-ohm crystals couldn't be used, if the other off-chip components are suitably adjusted.

Should we recommend values for the other off-chip components? Should we do it for 50-ohm crystals or 30-ohm crystals? With respect to what should we optimize their selection? Should we minimize start-up time or maximize frequency stability? In many applications, neither start-up time nor frequency stability are particularly critical, and our "recommendations" are only restricting your system to unnecessary tolerances. It all depends on the application.

Although we will neither "specify" nor "recommend" specific off-chip components, we do offer assistance in these tasks. Intel applications engineers are available to provide whatever technical assistance may be needed or desired by our customers in designing with Intel products.

This Application Note is intended to provide such assistance

in the design of oscillator circuits for microcontroller systems. Its purpose is to describe in a practical manner how oscillators work, how crystals and ceramic resonators work (and thus how to spec them), and what the on-chip amplifier looks like electronically and what its operating characteristics are. A BASIC program is provided in Appendix II to assist the designer in determining the effects of changing individual parameters. Suggestions are provided for establishing a pre-production test program.

## FEEDBACK OSCILLATORS

### Loop Gain

Figure 1 shows an amplifier whose output line goes into some passive network. If the input signal to the amplifier is  $v_1$ , then the output signal from the amplifier is  $v_2 = Av_1$  and the output signal from the passive network is  $v_3 = \beta v_2 = \beta Av_1$ . Thus  $\beta A$  is the overall gain from terminal 1 to terminal 3.

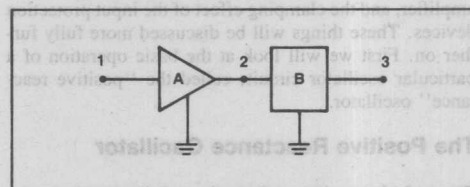


Figure 1 — Factors in Loop Gain

Now connect terminal 1 to terminal 3, so that the signal path forms a loop: 1 to 2 to 3, which is also 1. Now we have a feedback loop, and the gain factor  $\beta A$  is called the *loop gain*.

Gain factors are complex numbers. That means they have a magnitude and a phase angle, both of which vary with frequency. When writing a complex number, one must specify both quantities, magnitude and angle. A number whose magnitude is 3, and whose angle is 45 degrees is commonly written this way:  $3/45^\circ$ . The number 1 is, in complex number notation,  $1/0^\circ$ , while  $-1$  is  $1/180^\circ$ .

By closing the feedback loop in Figure 0, we force the equality

$$v_1 = \beta Av_1$$

This equation has two solutions:

- 1)  $v_1 = 0$ ;
- 2)  $\beta A = 1/0^\circ$ .

In a given circuit, either or both of the above solutions may be in effect. In the first solution the circuit is quiescent (no output signal). If you're trying to make an



oscillator, a no-signal condition is unacceptable. There are ways to guarantee that the second solution is the one that will be in effect, and that the quiescent condition will be excluded.

### How Feedback Oscillators Work

A feedback oscillator amplifies its own noise and feeds it back to itself in exactly the right phase, at the oscillation frequency, to build up and reinforce the desired oscillations. Its ability to do that depends on its loop gain. First, oscillations can occur only at the frequency for which the loop gain has a phase angle of 0 degrees. Second, build-up of oscillations will occur only if the loop gain exceeds 1 at that frequency. Build-up continues until nonlinearities in the circuit reduce the average value of the loop gain to exactly 1.

Start-up characteristics depend on the small-signal properties of the circuit, specifically, the small-signal loop gain. Steady-state characteristics of the oscillator depend on the large-signal properties of the circuit, such as the transfer curve (output voltage vs. input voltage) of the amplifier, and the clamping effect of the input protection devices. These things will be discussed more fully further on. First we will look at the basic operation of a particular oscillator circuit, called the "positive reactance" oscillator.

### The Positive Reactance Oscillator

Figure 2 shows the configuration of the positive reactance oscillator. The inverting amplifier, working into the impedance of the feedback network, produces an output signal that is nominally 180 degrees out of phase with its input. The feedback network must provide an additional 180 degrees phase shift, such that the overall loop gain has zero (or 360) degrees phase shift at the oscillation frequency.

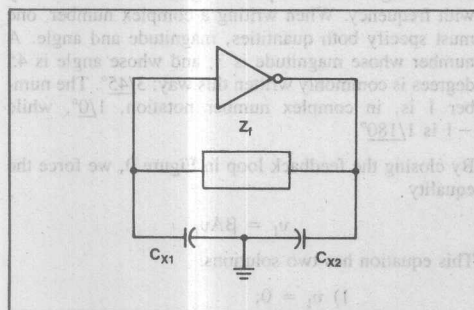


Figure 2 — Positive Reactance Oscillator

In order for the loop gain to have zero phase angle it is necessary that the feedback element  $Z_f$  have a positive

reactance. That is, it must be inductive. Then, the frequency at which the phase angle is zero is approximately the frequency at which

$$X_f = \frac{+1}{\omega C}$$

where  $X_f$  is the reactance of  $Z_f$  (the total  $Z_f$  being  $R_f + jX_f$ , and  $C$  is the series combination of  $C_{x1}$  and  $C_{x2}$ ).

$$C = \frac{C_{x1}C_{x2}}{C_{x1} + C_{x2}}$$

In other words,  $Z_f$  and  $C$  form a parallel resonant circuit.

If  $Z_f$  is an inductor, then  $X_f = \omega L$ , and the frequency at which the loop gain has zero phase is the frequency at which

$$\omega L = \frac{1}{\omega C}$$

or

$$\omega = \frac{1}{\sqrt{LC}}$$

Normally,  $Z_f$  is not an inductor, but it must still have a positive reactance in order for the circuit to oscillate. There are some piezoelectric devices on the market that show a positive reactance, and provide a more stable oscillation frequency than an inductor will. Quartz crystals can be used where the oscillation frequency is critical, and lower cost ceramic resonators can be used where the frequency is less critical.

When the feedback element is a piezoelectric device, this circuit configuration is called a Pierce oscillator. The advantage of piezoelectric resonators lies in their property of providing a wide range of positive reactance values over a very narrow range of frequencies. The reactance will equal  $1/\omega C$  at some frequency within this range, so the oscillation frequency will be within the same range. Typically, the width of this range is only .3% of the nominal frequency of a quartz crystal, and about 3% of the nominal frequency of a ceramic resonator. With relatively little design effort, frequency accuracies of .03% or better can be obtained with quartz crystals, and .3% or better with ceramic resonators.

### QUARTZ CRYSTALS

The crystal resonator is a thin slice of quartz sandwiched between two electrodes. Electrically, the device looks pretty much like a 5 or 6 pF capacitor, except that over certain ranges of frequencies the crystal has a positive (i.e., inductive) reactance.

The ranges of positive reactance originate in the piezoelectric property of quartz: Squeezing the crystal gener-

ates an internal E-field. The effect is reversible: Applying an E-field causes a mechanical deflection. Applying an AC E-field causes the crystal to vibrate. At certain vibrational frequencies there is a mechanical resonance. As the E-field frequency approaches a frequency of mechanical resonance, the measured reactance of the crystal becomes positive, as shown in Figure 3.

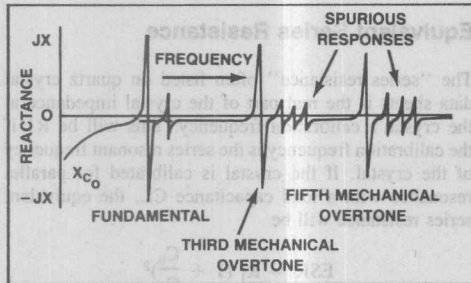


Figure 3 — Crystal Reactance vs. Frequency

Typically there are several ranges of frequencies wherein the reactance of the crystal is positive. Each range corresponds to a different mode of vibration in the crystal. The main resonances are the so-called fundamental response and the third and fifth overtone responses.

The overtone responses shouldn't be confused with the harmonics of the fundamental. They're not harmonics, but different vibrational modes. They're not in general at exact integer multiples of the fundamental frequency. There will also be "spurious" responses, occurring typically a few hundred KHz above each main response.

To assure that an oscillator starts in the desired mode on power-up, something must be done to suppress the loop gain in the undesired frequency ranges. The crystal itself provides some protection against unwanted modes of oscillation; too much resistance in that mode, for example. Additionally, junction capacitances in the amplifying devices tend to reduce the gain at higher frequencies, and thus may discriminate against unwanted modes. In some cases a circuit fix is necessary, such as inserting a trap, a phase shifter, or ferrite beads to kill oscillations in unwanted modes.

### Crystal Parameters

#### Equivalent Circuit

Figure 4 shows an equivalent circuit that is used to represent the crystal for circuit analysis.

The  $R_1$ - $L_1$ - $C_1$  branch is called the motional arm of the crystal. The values of these parameters derive from the mechanical properties of the crystal and are constant for a given mode of vibration. Typical values for various nominal frequencies are shown in Table 1.

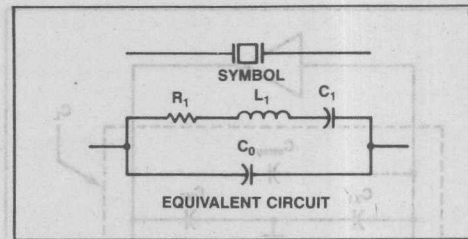


Figure 4 — Quartz Crystal:  
Symbol and Equivalent Circuit

$C_0$  is called the shunt capacitance of the crystal. This is the capacitance of the crystal's electrodes and the mechanical holder. If one were to measure the reactance of the crystal at a frequency far removed from a resonance frequency, it is the reactance of this capacitance that would be measured. It's normally 3 to 7 pF.

Table 1 — Typical Crystal Parameters

frequency MHz	$R_1$ ohms	$L_1$ mH	$C_1$ pF	$C_0$ pF
2	100	520	.012	4
4.608	36	117	.010	2.9
11.25	19	8.38	.024	5.4

The series resonant frequency of the crystal is the frequency at which  $L_1$  and  $C_1$  are in resonance. This frequency is given by

$$f_s = \frac{1}{2\pi\sqrt{L_1 C_1}}$$

At this frequency the impedance of the crystal is  $R_1$  in parallel with the reactance of  $C_0$ . For most purposes, this impedance is taken to be just  $R_1$ , since the reactance of  $C_0$  is so much larger than  $R_1$ .

### Load Capacitance

A crystal oscillator circuit such as the one shown in Figure 2 (redrawn in Figure 5) operates at the frequency for which the crystal is antiresonant (ie, parallel-resonant) with the total capacitance across the crystal terminals external to the crystal. This total capacitance external to the crystal is called the load capacitance.

As shown in Figure 5, the load capacitance is given by

$$C_L = \frac{C_{X1} C_{X2}}{C_{X1} + C_{X2}} + C_{\text{stray}}$$

The crystal manufacturer needs to know the value of  $C_L$  in order to adjust the crystal to the specified frequency.

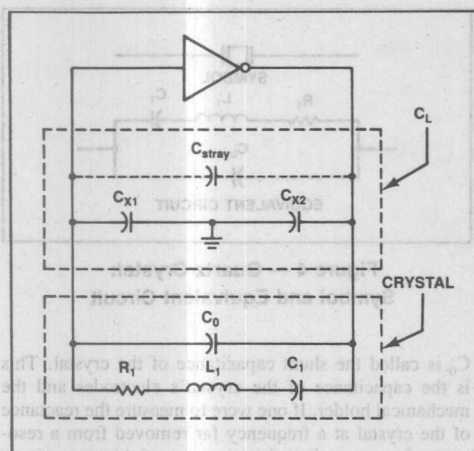


Figure 5 — Load Capacitance

The adjustment involves putting the crystal in *series* with the specified  $C_L$ , and then "trimming" the crystal to obtain resonance of the series combination of the crystal and  $C_L$  at the specified frequency. Because of the high  $Q$  of the crystal, the resonant frequency of the series combination of the crystal and  $C_L$  is the same as the antiresonant frequency of the *parallel* combination of the crystal and  $C_L$ . This frequency is given by

$$f_a = \frac{1}{2\pi\sqrt{L_1 C_1 (C_L + C_0) / (C_1 + C_L + C_0)}}$$

These frequency formulas are derived (in Appendix I) from the equivalent circuit of the crystal, using the assumptions that the  $Q$  of the crystal is extremely high, and that the circuit external to the crystal has no effect on the frequency other than to provide the load capacitance  $C_L$ . The latter assumption is not precisely true, but it is close enough for present purposes.

### "Series" vs. "Parallel" Crystals

There is no such thing as a "series cut" crystal as opposed to a "parallel cut" crystal. There are different cuts of crystal, having to do with the parameters of its motional arm in various frequency ranges, but there is no special cut for series or parallel operation.

An oscillator is series resonant if the oscillation frequency is  $f_s$  of the crystal. To operate the crystal at  $f_s$ , the amplifier has to be noninverting. When buying a crystal for such an oscillator, one does not specify a load capacitance. Rather, one specifies the loading condition as "series."

If a "series" crystal is put into an oscillator that has an inverting amplifier, it will oscillate in parallel resonance with the load capacitance presented to the crystal by the

oscillator circuit, at a frequency slightly above  $f_s$ . In fact, at approximately

$$f_a = f_s \left( 1 + \frac{C_1}{2(C_L + C_0)} \right)$$

This frequency would typically be about 0.02% above  $f_s$ .

### Equivalent Series Resistance

The "series resistance" often listed on quartz crystal data sheets is the real part of the crystal impedance at the crystal's calibration frequency. This will be  $R_1$  if the calibration frequency is the series resonant frequency of the crystal. If the crystal is calibrated for parallel resonance with a load capacitance  $C_L$ , the equivalent series resistance will be

$$ESR = R_1 \left( 1 + \frac{C_0}{C_L} \right)^2$$

The crystal manufacturer measures this resistance at the calibration frequency during the same operation in which the crystal is adjusted to the calibration frequency.

### Frequency Tolerance

Frequency tolerance as discussed here is not a requirement on the crystal, but on the complete oscillator. There are two types of frequency tolerances on oscillators: frequency *accuracy* and frequency *stability*. Frequency accuracy refers to the oscillator's ability to run at an exact specified frequency. Frequency stability refers to the constancy of the oscillation frequency.

Frequency accuracy requires mainly that the oscillator circuit present to the crystal the same load capacitance that it was adjusted for. Frequency stability requires mainly that the load capacitance be constant.

In most digital applications the accuracy and stability requirements on the oscillator are so wide that it makes very little difference what load capacitance the crystal was adjusted to, or what load capacitance the circuit actually presents to the crystal. For example, if a crystal was calibrated to a load capacitance of 25 pF, and is used in a circuit whose actual load capacitance is 50 pF, the frequency error on that account would be less than 0.01%.

In a positive reactance oscillator, the crystal only needs to be in the intended response mode for the oscillator to satisfy a 0.5% or better frequency tolerance. That's because for any load capacitance the oscillation frequency is certain to be between the crystal's resonant and antiresonant frequencies.

Phase shifts that take place within the amplifier part of the oscillator will also affect frequency accuracy and



stability. These phase shifts can normally be modeled as an "output capacitance" that, in the positive reactance oscillator, parallels  $C_{X2}$ . The predictability and constancy of this output capacitance over temperature and device sample will be the limiting factor in determining the tolerances that the circuit is capable of holding.

### Drive Level

Drive level refers to the power dissipation in the crystal. There are two reasons for specifying it. One is that the parameters in the equivalent circuit are somewhat dependent on the drive level at which the crystal is calibrated. The other is that if the application circuit exceeds the test drive level by too much, the crystal may be damaged. Note that the terms "test drive level" and "rated drive level" both refer to the drive level at which the crystal is calibrated. Normally, in a microcontroller system, neither the frequency tolerances nor the power levels justify much concern for this specification. Some crystal manufacturers don't even require it for microprocessor crystals.

In a positive reactance oscillator, if one assumes the peak voltage across the crystal to be something in the neighborhood of  $V_{CC}$ , the power dissipation can be approximated as

$$P = 2R_1[\pi f (C_L + C_0) V_{CC}]^2$$

This formula is derived in Appendix I. In a 5V system,  $P$  rarely evaluates to more than a milliwatt. Crystals with a standard 1 or 2 mW drive level rating can be used in most digital systems.

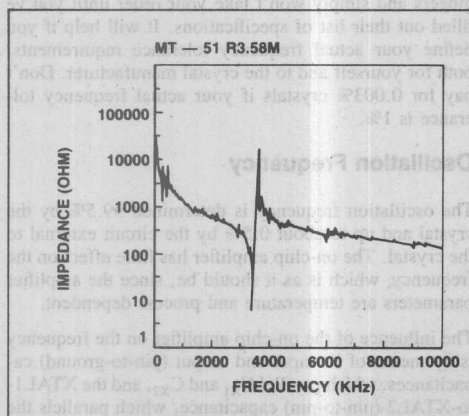


Figure 6 — Ceramic Resonator Impedance vs. Frequency (Test Data Supplied by NTK Technical Ceramics)

## CERAMIC RESONATORS

Ceramic resonators operate on the same basic principles as a quartz crystal. Like quartz crystals, they are piezoelectric, have a reactance versus frequency curve similar to a crystal's, and an equivalent circuit that looks just like a crystal's (with different parameter values, however).

The frequency tolerance of a ceramic resonator is about two orders of magnitude wider than a crystal's, but the ceramic is somewhat cheaper than a crystal. It may be noted for comparison that quartz crystals with relaxed tolerances cost about twice as much as ceramic resonators. For purposes of clocking a microcontroller, the frequency tolerance is often relatively noncritical, and the economic consideration becomes the dominant factor.

Figure 6 shows a graph of impedance magnitude versus frequency for a 3.58MHz ceramic resonator. (Note that Figure 6 is a graph of  $|Z_f|$  versus frequency, whereas Figure 3 is a graph of  $X_f$  versus frequency.) A number of spurious responses are apparent in Figure 6. The manufacturers state that spurious responses are more prevalent in the lower frequency resonators (kHz range) than in the higher frequency units (MHz range). For our purposes only the MHz range ceramics need to be considered.

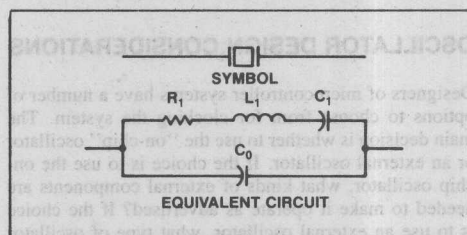


Figure 7 — Ceramic Resonator: Symbol and Equivalent Circuit

Figure 7 shows the symbol and equivalent circuit for the ceramic resonator, both of which are the same as for the crystal. The parameters have different values, however, as listed in Table 2.

Table 2 — Typical Ceramic Parameters

frequency MHz	$R_1$ ohms	$L_1$ mH	$C_1$ pF	$C_0$ pF
3.58	7	.113	19.6	140
6.0	8	.094	8.3	60
8.0	7	.092	4.6	40
11.0	10	.057	3.9	30



Note that the motional arm of the ceramic resonator tends to have less resistance than the quartz crystal and also a vastly reduced  $L_1/C_1$  ratio. This results in the motional arm having a  $Q$  (given by  $(1/R_1) \sqrt{L_1/C_1}$ ) that is typically two orders of magnitude lower than that of a quartz crystal. The lower  $Q$  makes for a faster start-up of the oscillator and for a less closely controlled frequency (meaning that circuitry external to the resonator will have more influence on the frequency than with a quartz crystal).

Another major difference is that the shunt capacitance of the ceramic resonator is an order of magnitude higher than  $C_0$  of the quartz crystal and more dependent on the frequency of the resonator.

The implications of these differences are not all obvious, but some will be indicated in the section on Oscillator Calculations.

### Specifications for Ceramic Resonators

Ceramic resonators are easier to specify than quartz crystals. All the vendor wants to know is the desired frequency and the chip you want it to work with. They'll supply the resonators, a circuit diagram showing the positions and values of other external components that may be required and a guarantee that the circuit will work properly at the specified frequency.

## OSCILLATOR DESIGN CONSIDERATIONS

Designers of microcontroller systems have a number of options to choose from for clocking the system. The main decision is whether to use the "on-chip" oscillator or an external oscillator. If the choice is to use the on-chip oscillator, what kinds of external components are needed to make it operate as advertised? If the choice is to use an external oscillator, what type of oscillator should it be?

The decisions have to be based on both economic and technical requirements. In this section we'll discuss some of the factors that should be considered.

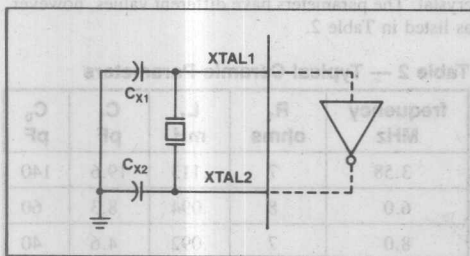


Figure 8 — Using the "On-Chip" Oscillator

## On-Chip Oscillators

In most cases, the on-chip amplifier with the appropriate external components provides the most economical solution to the clocking problem. Exceptions may arise in severe environments when frequency tolerances are tighter than about 0.01%.

The external components that need to be added are a positive reactance (normally a crystal or ceramic resonator) and the two capacitors  $C_{X1}$  and  $C_{X2}$ , as shown in Figure 8.

### Crystal Specifications

Specifications for an appropriate crystal are not very critical, unless the frequency is. Any fundamental-mode crystal of medium or better quality can be used.

We are often asked what maximum crystal resistance should be specified. The best answer to this question is the lower the better, but use what's available. The crystal resistance will have some effect on start-up time and steady-state amplitude, but not so much that it can't be compensated for by appropriate selection of the capacitances  $C_{X1}$  and  $C_{X2}$ .

Similar questions are asked about specifications of load capacitance and shunt capacitance. The best advice we can give is to understand what these parameters mean and how they affect the operation of the circuit (that being the purpose of this Application Note), and then decide for yourself if such specifications are meaningful in your application or not. Normally, they're not, unless your frequency tolerances are tighter than about 0.1%.

Part of the problem is that crystal manufacturers are accustomed to talking "ppm" tolerances with radio engineers and simply won't take your order until you've filled out their list of specifications. It will help if you define your actual frequency tolerance requirements, both for yourself and to the crystal manufacturer. Don't pay for 0.003% crystals if your actual frequency tolerance is 1%.

### Oscillation Frequency

The oscillation frequency is determined 99.5% by the crystal and up to about 0.5% by the circuit external to the crystal. The on-chip amplifier has little effect on the frequency, which is as it should be, since the amplifier parameters are temperature and process dependent.

The influence of the on-chip amplifier on the frequency is by means of its input and output (pin-to-ground) capacitances, which parallel  $C_{X1}$  and  $C_{X2}$ , and the XTAL1-to-XTAL2 (pin-to-pin) capacitance, which parallels the crystal. The input and pin-to-pin capacitances are about 7 pF each. Internal phase deviations from the nominal 180° can be modeled as an output capacitance of 25 to 30 pF. These deviations from the ideal have less effect

in the positive reactance oscillator (with the inverting amplifier) than in a comparable series resonant oscillator (with the noninverting amplifier) for two reasons: first, the effect of the output capacitance is lessened, if not swamped, by the off-chip capacitor; secondly, the positive reactance oscillator is less sensitive, frequency-wise, to such phase errors.

### Selection of $C_{X1}$ and $C_{X2}$

Optimal values for the capacitors  $C_{X1}$  and  $C_{X2}$  depend on whether a quartz crystal or ceramic resonator is being used, and also on application-specific requirements on start-up time and frequency tolerance.

Start-up time is sometimes more critical in microcontroller systems than frequency stability, because of various reset and initialization requirements.

Less commonly, accuracy of the oscillator frequency is also critical, for example, when the oscillator is being used as a time base. As a general rule, fast start-up and stable frequency tend to pull the oscillator design in opposite directions.

Considerations of both start-up time and frequency stability over temperature suggest that  $C_{X1}$  and  $C_{X2}$  should be about equal and at least 20 pF. (But they don't have to be either.) Increasing the value of these capacitances above some 40 or 50 pF improves frequency stability. It also tends to increase the start-up time. There is a maximum value (several hundred pF, depending on the value of  $R_1$  of the quartz or ceramic resonator) above which the oscillator won't start up at all.

If the on-chip amplifier is a simple inverter, such as in the 8051, the user can select values for  $C_{X1}$  and  $C_{X2}$  between some 20 and 100 pF, depending on whether start-up time or frequency stability is the more critical parameter in a specific application. If the on-chip amplifier is a Schmitt Trigger, such as in the 8048, smaller values of  $C_{X1}$  must be used (5 to 30 pF), in order to prevent the oscillator from running in a relaxation mode.

Later sections in this Application Note will discuss the effects of varying  $C_{X1}$  and  $C_{X2}$  (as well as other parameters), and will have more to say on their selection.

### Placement of Components

Noise glitches arriving at the XTAL1 or XTAL2 pins at the wrong time can cause a miscount in the internal clock-generating circuitry. These kinds of glitches can be produced through capacitive coupling between the oscillator components and PCB traces carrying digital signals with fast rise and fall times. For this reason, the oscillator components should be mounted close to the chip and have short, direct traces to the XTAL1, XTAL2, and VSS pins.

### Clocking Other Chips

There are times when it would be desirable to use the on-chip oscillator to clock other chips in the system.

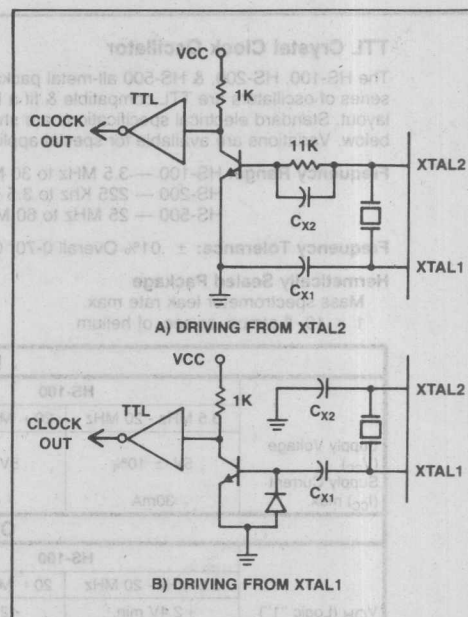


Figure 9 — Using the On-Chip Oscillator to Drive Other Chips

This can be done if an appropriate buffer is used. A TTL buffer puts too much load on the on-chip amplifier for reliable start-up. A CMOS buffer (such as the 74HC04) can be used, if it's fast enough and if its  $V_{IH}$  and  $V_{IL}$  specs are compatible with the available signal amplitudes. Circuits such as shown in Figure 9 might also be considered for these types of applications.

Clock-related signals are available at the TO pin in the MCS-48 products, at ALE in the MCS-48 and MCS-51 lines, and the iACX-96 controllers provide a CLKOUT signal.

### External Oscillators

When technical requirements dictate the use of an external oscillator, the external drive requirements for the microcontroller, as published in the data sheet, must be carefully noted. The logic levels are not in general TTL-compatible. And each controller has its idiosyncracies in this regard. The 8048, for example, requires that both XTAL1 and XTAL2 be driven. The 8051 can be driven that way, but the data sheet suggests the simpler method of grounding XTAL1 and driving XTAL2. For this method, the driving source must be capable of sinking some current when XTAL2 is being driven low.

For the external oscillator itself, there are basically two choices: ready-made and home-grown.

### TTL Crystal Clock Oscillator

The HS-100, HS-200, & HS-500 all-metal package series of oscillators are TTL compatible & fit a DIP layout. Standard electrical specifications are shown below. Variations are available for special applications.

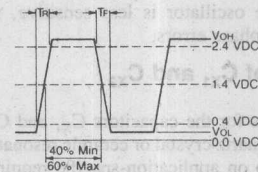
**Frequency Range:** HS-100 — 3.5 MHz to 30 MHz  
 HS-200 — 225 KHz to 3.5 MHz  
 HS-500 — 25 MHz to 60 MHz

**Frequency Tolerance:**  $\pm .01\%$  Overall 0-70° C

#### Hermetically Sealed Package

Mass spectrometer leak rate max.  
 $1 \times 10^{-8}$  atmos. cc/sec. of helium

#### OUTPUT WAVE FORM



INPUT				
	HS-100		HS-200	HS-500
	3.5 MHz - 20 MHz	20 + MHz - 30 MHz	225 KHz - 4.0 MHz	25 MHz - 60 MHz
Supply Voltage (V <sub>CC</sub> )	5V $\pm$ 10%	5V $\pm$ 10%	5V $\pm$ 10%	5V $\pm$ 5%
Supply Current (I <sub>CC</sub> ) max.	30mA	40mA	85mA	50mA
OUTPUT				
	HS-100	HS-200	HS-500	
	3.5 MHz - 20 MHz	20 + MHz - 30 MHz	225 KHz - 4.0 MHz	25 MHz - 60 MHz
V <sub>OH</sub> (Logic "1")	+ 2.4V min. <sup>1</sup>	+ 2.7V min. <sup>2</sup>	+ 2.4V min. <sup>1</sup>	+ 2.7V min. <sup>2</sup>
V <sub>OL</sub> (Logic "0")	+ 0.4V max. <sup>3</sup>	+ 0.5V max. <sup>4</sup>	+ 0.4V max. <sup>3</sup>	+ 0.5V max. <sup>4</sup>
Symmetry	60-40% <sup>5</sup>	60-40% <sup>5</sup>	55-45% <sup>5</sup>	60-40% <sup>5</sup>
T <sub>R</sub> , T <sub>F</sub> (Rise & Fall Time)	< 10ns <sup>6</sup>	< 5ns <sup>6</sup>	15ns <sup>6</sup>	5ns <sup>6</sup>
Output Short Circuit Current	18mA min.	40mA min.	18mA min.	40mA min.
Output Load	1 to 10 TTL Loads <sup>7</sup>	1 to 10 TTL Loads <sup>8</sup>	1 to 10 TTL Loads <sup>7</sup>	1 to 10 TTL Loads <sup>8</sup>
<b>CONDITIONS</b> <sup>1</sup> I <sub>O</sub> source — 400 $\mu$ A max. <sup>4</sup> I <sub>O</sub> sink — 20.0mA max. <sup>7</sup> 1.6mA per load <sup>2</sup> I <sub>O</sub> source — 1.0mA max. <sup>5</sup> V <sub>O</sub> — 1.4V <sup>8</sup> 2.0mA per load <sup>3</sup> I <sub>O</sub> sink — 16.0mA max. <sup>6</sup> (0.4V to 2.4V)				

Figure 10 — Pre-packaged Oscillator Data\*

Prepackaged oscillators are available from most crystal manufacturers, and have the advantage that the system designer can treat the oscillator as a black box whose performance is guaranteed by people who carry many years of experience in designing and building oscillators. Figure 10 shows a typical data sheet for some prepackaged oscillators. Oscillators are also available with complementary outputs.

If the oscillator is to drive the microcontroller directly, one will want to make a careful comparison between the external drive requirements in the microcontroller data sheet and the oscillator's output logic levels and test conditions.

If oscillator stability is less critical than cost, the user

may prefer to go with an in-house design. Not without some precautions, however.

It's easy to design oscillators that work. Almost all of them do work, even if the designer isn't too clear on why. The key point here is that *almost* all of them work. The problems begin when the system goes into production, and marginal units commence malfunctioning in the field. Most digital designers, after all, are not very adept at designing oscillators for production.

Oscillator design is somewhat of a black art, with the quality of the finished product being very dependent on the designer's experience and intuition. For that reason the most important consideration in any design is to have an adequate preproduction test program. Preproduction tests are discussed later in this Application Note. Here we will discuss some of the design options and take a look at some commonly used configurations.

\*Reprinted with the permission of © Midland-Ross Corporation 1982

## Gate Oscillators versus Discrete Devices

Digital systems designers are understandably reluctant to get involved with discrete devices and their peculiarities (biasing techniques, etc.). Besides, the component count for these circuits tends to be quite a bit higher than what a digital designer is used to seeing for that amount of functionality. Nevertheless, if there are unusual requirements on the accuracy and stability of the clock frequency, it should be noted that discrete device oscillators can be tailored to suit the exact needs of the application and perfected to a level that would be difficult for a gate oscillator to approach.

In most cases, when an external oscillator is needed, the designer tends to rely on some form of a gate oscillator. A TTL inverter with a resistor connecting the output to the input makes a suitable inverting amplifier. The resistor holds the inverter in the transition region between logical high and low, so that at least for start-up purposes the inverter is a linear amplifier.

The feedback resistance has to be quite low, however, since it must conduct current sourced by the input pin without allowing the DC input voltage to get too far above the DC output voltage. For biasing purposes, the feedback resistance should not exceed a few k-ohms.

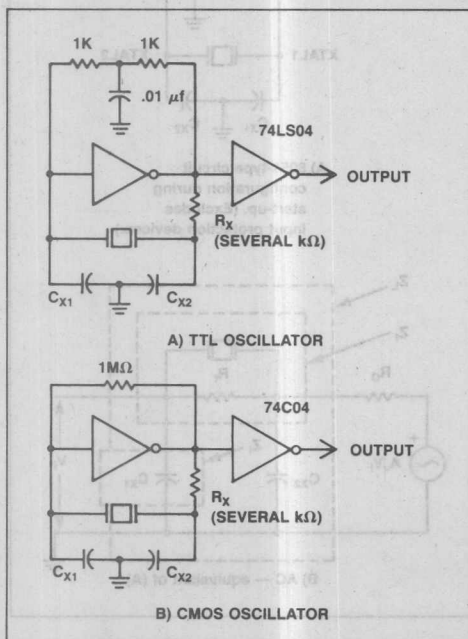


Figure 11 — Commonly Used Gate Oscillators

But shunting the crystal with such a low resistance does not encourage start-up.

Consequently, the configuration in Figure 11A might be suggested. By breaking  $R_f$  into two parts and AC-grounding the midpoint, one achieves the DC feedback required to hold the inverter in its active region, but without the negative signal feedback that is in effect telling the circuit *not* to oscillate. However, this biasing scheme will increase the start-up time, and relaxation-type oscillations are also possible.

A CMOS inverter, such as the 74HC04, might work better in this application, since a larger  $R_f$  can be used to hold the inverter in its linear region.

Logic gates tend to have a fairly low output resistance, which destabilizes the oscillator. For that reason a resistor  $R_x$  is often added to the feedback network, as shown in Figure 11 A and B. At higher frequencies a 20 or 30 pF capacitor is sometimes used in the  $R_x$  position, to compensate for some of the internal propagation delay.

Reference 1 contains an excellent discussion of gate oscillators, and a number of design examples.

## Fundamental versus Overtone Operation

It's easier to design an oscillator circuit to operate in the resonator's fundamental response mode than to design one for overtone operation. A quartz crystal whose fundamental response mode covers the desired frequency can be obtained up to some 30 MHz. For frequencies above that, the crystal might be used in an overtone mode.

Several problems arise in the design of an overtone oscillator. One is to stop the circuit from oscillating in the fundamental mode, which is what it would really rather do, for a number of reasons, involving both the amplifying device and the crystal. An additional problem with overtone operation is an increased tendency to spurious oscillations. That is because the  $R_f$  of various spurious modes is likely to be about the same as  $R_f$  of the intended overtone response. It may be necessary, as suggested in reference 1, to specify a "spurious-to-main-response" resistance ratio to avoid the possibility of trouble.

Overtone oscillators are not to be taken lightly. One would be well advised to consult with an engineer who is knowledgeable in the subject during the design phase of such a circuit.

## Series versus Parallel Operation

Series resonant oscillators use noninverting amplifiers. To make a noninverting amplifier out of logic gates requires that two inverters be used, as shown in Figure 12.

This type of circuit tends to be inaccurate and unstable



in frequency over variations in temperature and VCC. It has a tendency to oscillate at overtones, and to oscillate through  $C_0$  of the crystal or some stray capacitance rather than as controlled by the mechanical resonance of the crystal.

The demon in series resonant oscillators is the phase shift in the amplifier. The series resonant oscillator wants more than just a "noninverting" amplifier — it wants a zero phase-shift amplifier. Multistage noninverting amplifiers tend to have a considerably lagging phase shift, such that the crystal reactance must be capacitive in order to bring the total phase shift around the feedback loop back up to 0. In this mode, a "12 MHz" crystal may be running at 8 or 9 MHz. One can put a capacitor in series with the crystal to relieve the crystal of having to produce all of the required phase shift, and bring the oscillation frequency closer to  $f_s$ . However, to further complicate the situation, the amplifier's phase shift is strongly dependent on frequency, temperature, VCC, and device sample.

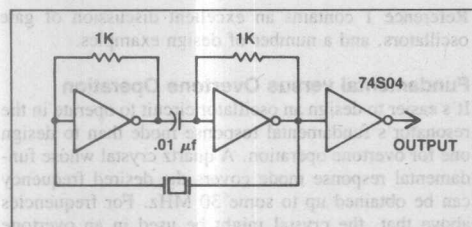


Figure 12 — "Series Resonant"  
Gate Oscillator

Positive reactance oscillators ("parallel resonant") use inverting amplifiers. A single logic inverter can be used for the amplifier, as in Figure 11. The amplifier's phase shift is less critical, compared to the series resonant circuit, and since only one inverter is involved there's less phase error anyway. The oscillation frequency is effectively bounded by the resonant and antiresonant frequencies of the crystal itself. In addition, the feedback network includes capacitors that parallel the input and output terminals of the amplifier, thus reducing the effect of unpredictable capacitances at these points.

## MORE ABOUT USING THE "ON-CHIP" OSCILLATORS

In this section we will describe the on-chip inverters on selected microcontrollers in some detail, and discuss criteria for selecting components to work with them. Future data sheets will supplement this discussion with updates and information pertinent to the use of each chip's oscillator circuitry.

## Oscillator Calculations

Oscillator design, though aided by theory, is still largely an empirical exercise. The circuit is inherently nonlinear, and the normal analysis parameters vary with instantaneous voltage. In addition, when dealing with the on-chip circuitry, we have FETs being used as resistors, resistors being used as interconnects, distributed delays, input protection devices, parasitic junctions, and processing variations.

Consequently, oscillator calculations are never very precise. They can be useful, however, if they will at least indicate the effects of variations in the circuit parameters on start-up time, oscillation frequency, and steady-state amplitude. Start-up time, for example, can be taken as an indication of start-up reliability. If preproduction tests indicate a possible start-up problem, a relatively inexperienced designer can at least be made aware of what

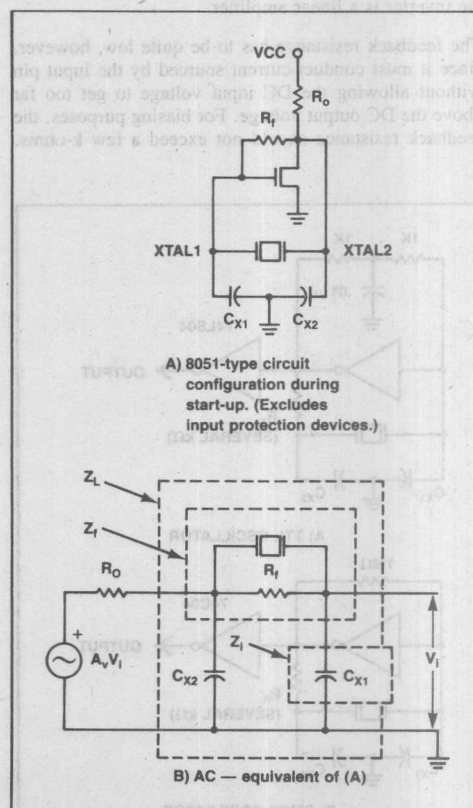


Figure 13 — Oscillator Circuit Model  
Used in Start-Up Calculations

parameter may be causing the marginality, and what direction to go in to fix it.

The analysis used here is mathematically straightforward but algebraically intractable. That means it's relatively easy to understand and program into a computer, but it will not yield a neat formula that gives, say, steady-state amplitude as a function of this or that list of parameters. A listing of a BASIC program that implements the analysis will be found in Appendix II.

When the circuit is first powered up, and before the oscillations have commenced (and if the oscillations fail to commence), the oscillator can be treated as a small signal linear amplifier with feedback. In that case, standard small-signal analysis techniques can be used to determine start-up characteristics. The circuit model used in this analysis is shown in Figure 13.

The circuit approximates that there are no high-frequency effects within the amplifier itself, such that its high-frequency behavior is dominated by the load impedance  $Z_L$ . This is a reasonable approximation for single-stage amplifiers of the type used in 8051-type devices. Then the gain of the amplifier as a function of frequency is

$$A = \frac{A_v Z_L}{Z_L + R_0}$$

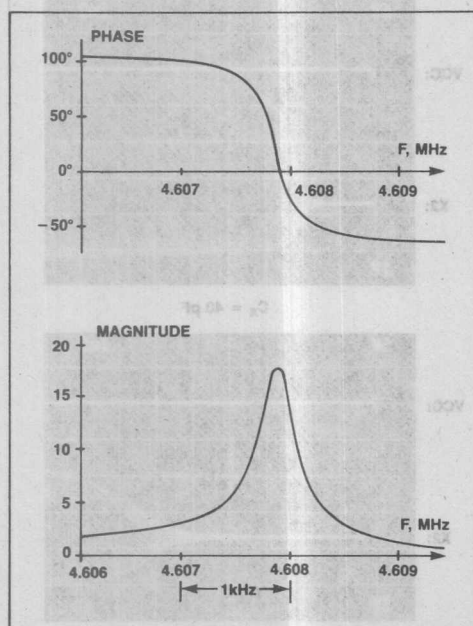


Figure 14 — Loop Gain versus Frequency (4.608 MHz Crystal)

The gain of the feedback network is

$$\beta = \frac{Z_i}{Z_i + Z_f}$$

And the loop gain is

$$\beta A = \frac{1}{Z_i + Z_f} \cdot \frac{A_v Z_L}{Z_L + R_0}$$

The impedances  $Z_L$ ,  $Z_f$ , and  $Z_i$  are defined in Figure 13B.

Figure 14 shows the way the loop gain thus calculated (using typical 8051-type parameters and a 4.608 MHz crystal) varies with frequency. The frequency of interest is the one for which the phase of the loop gain is zero. The accepted criterion for start-up is that the magnitude of the loop gain must exceed unity at this frequency. This is the frequency at which the circuit is in resonance. It corresponds very closely with the antiresonant frequency of the motional arm of the crystal in parallel with  $C_L$ .

Figure 15 shows the way the loop gain varies with frequency when the parameters of a 3.58MHz ceramic resonator are used in place of a crystal (the amplifier parameters being typical 8051, as in Figure 14). Note the different frequency scales.

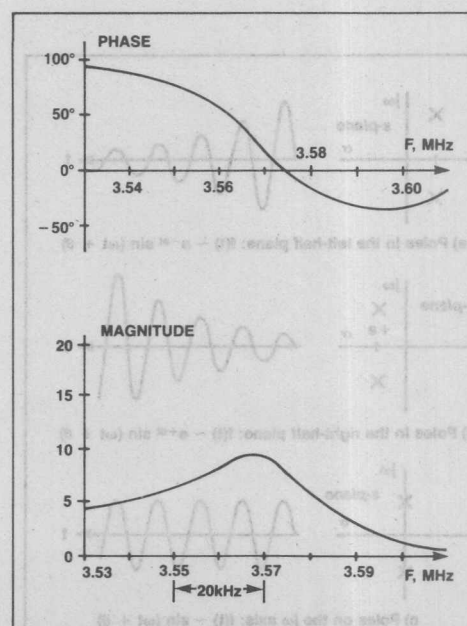


Figure 15 — Loop Gain versus Frequency (3.58MHz Ceramic)

## Start-Up Characteristics

It is common, in studies of feedback systems, to examine the behavior of the closed loop gain as a function of complex frequency  $s = \sigma + j\omega$ ; specifically, to determine the location of its poles in the complex plane. A pole is a point on the complex plane where the gain function goes to infinity. Knowledge of its location can be used to predict the response of the system to an input disturbance.

The way that the response function depends on the location of the poles is shown in Figure 16. Poles in the left half plane cause the response function to take the form of a damped sinusoid. Poles in the right half plane cause the response function to take the form of an exponentially growing sinusoid. In general,

$$v(t) \sim e^{at} \sin(\omega t + \theta)$$

where  $a$  is the real part of the pole frequency. Thus if the pole is in the right half plane,  $a$  is positive and the sinusoid grows. If the pole is in the left half plane,  $a$  is negative and the sinusoid is damped.

The same type of analysis can usefully be applied to oscillators. In this case, however, rather than trying to ensure that the poles are in the left half plane, we would seek to ensure that they're in the *right* half plane. An

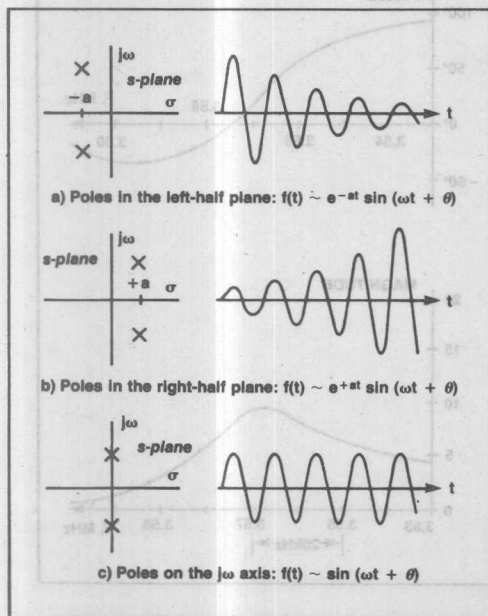


Figure 16 — Do You Know Where Your Poles are Tonight?

exponentially growing sinusoid is exactly what is wanted from an oscillator that has just been powered up.

The gain function of interest in oscillators is  $1/(1 - \beta A)$ . Its poles are at the complex frequencies where  $\beta A = 1 \angle 0^\circ$ , because that value of  $\beta A$  causes the gain function to go to infinity. The oscillator will start up if the real part of the pole frequency is positive. More importantly, the *rate* at which it starts up is indicated by how *much* greater than 0 the real part of the pole frequency is.

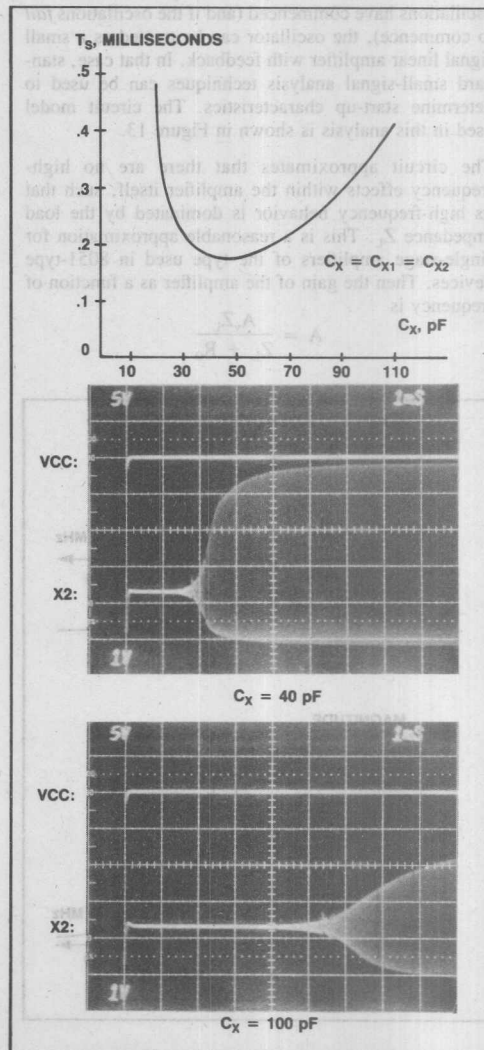


Figure 17 — Oscillator Start-Up (4.608 MHz Crystal from Standard Crystal Corp.)

The circuit in Figure 13B can be used to find the pole frequencies of the oscillator gain function. All that needs to be done is evaluate the impedances at complex frequencies  $\sigma + j\omega$  rather than just at  $\omega$ , and find the value of  $\sigma + j\omega$  for which  $BA = 1 \angle 0^\circ$ . The larger that value of  $\sigma$  is, the faster the oscillator will start up.

Of course, other things besides pole frequencies, things like the VCC rise time, are at work in determining the start-up time. But to the extent that the pole frequencies

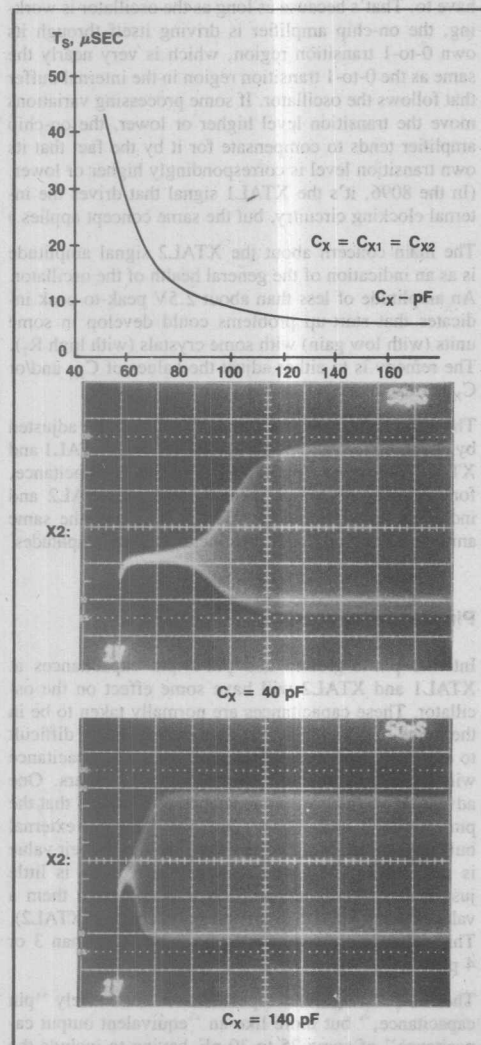


Figure 18 — Oscillator Start-Up (3.58 MHz Ceramic Resonator from NTK Technical Ceramics.)

do affect start-up time, we can obtain results like those in Figures 17 and 18.

To obtain these figures, the pole frequencies were computed for various values of capacitance  $C_X$  from XTAL1 and XTAL2 to ground (thus  $C_{X1} = C_{X2} = C_X$ ). Then a "time constant" for start-up was calculated as

$T_s = \frac{1}{\sigma}$  where  $\sigma$  is the real part of the pole frequency (rad/sec), and this time constant is plotted versus  $C_X$ .

A short time constant means faster start-up. A long time constant means slow start-up. Observations of actual start-ups are shown in the figures. Figure 17 is for a typical 8051 with a 4.608 MHz crystal supplied by Standard Crystal Corp., and Figure 18 is for a typical 8051 with a 3.58MHz ceramic resonator supplied by NTK Technical Ceramics, Ltd.

It can be seen in Figure 17 that, for this crystal, values of  $C_X$  between 30 and 50 pF minimize start-up time, but that the exact value in this range is not particularly important, even if the start-up time itself is critical.

As previously mentioned, start-up time can be taken as an indication of start-up reliability. Start-up problems are normally associated with  $C_{X1}$  and  $C_{X2}$  being too small or too large for a given resonator. If the parameters of the resonator are known, curves such as in Figure 17 or 18 can be generated to define acceptable ranges of values for these capacitors.

As the oscillations grow in amplitude, they reach a level at which they undergo severe clipping within the amplifier, in effect reducing the amplifier gain. As the amplifier gain decreases, the poles move towards the  $j\omega$  axis. In steady-state, the poles are on the  $j\omega$  axis and the amplitude of the oscillations is constant.

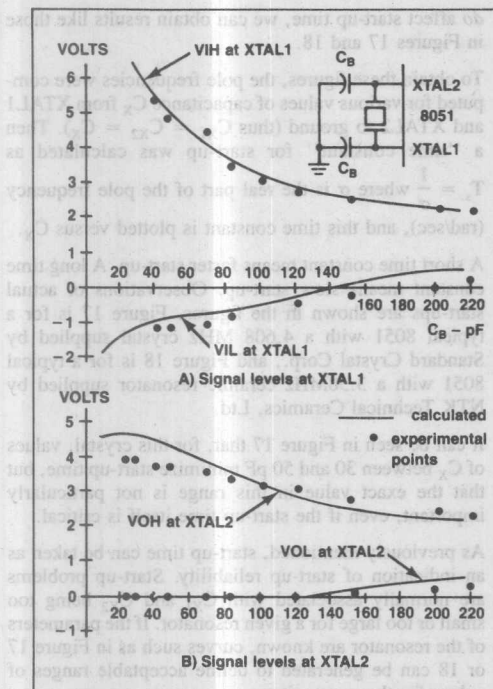
### Steady-State Characteristics

Steady-state analysis is greatly complicated by the fact that we are dealing with large signals and nonlinear circuit response. The circuit parameters vary with instantaneous voltage, and a number of clamping and clipping mechanisms come into play. Analyses that take all these things into account are too complicated to be of general use, and analyses that don't take them into account are too inaccurate to justify the effort.

There is a steady-state analysis in Appendix II that takes some of the complications into account and ignores others. Figure 19 shows the way the steady-state amplitudes thus calculated (using typical 8051 parameters and a 4.608 MHz crystal) vary with equal bulk capacitance placed from XTAL1 and XTAL2 to ground. Experimental results are shown for comparison.

The waveform at XTAL1 is a fairly clean sinusoid. Its negative peak is normally somewhat below zero, at a level which is determined mainly by the input protection circuitry at XTAL1.





**Figure 19 — Calculated and Experimental Steady-State Amplitudes vs. Bulk Capacitance from XTAL1 and XTAL2 to ground.**

The input protection circuitry consists of an ohmic resistor and an enhancement-mode FET with the gate and source connected to ground (VSS), as shown in Figure 20 for the 8051, and in Figure 21 for the 8048. Its function is to limit the positive voltage at the gate of the input FET to the avalanche voltage of the drain junction. If the input pin is driven below VSS, the drain and source of the protection FET interchange roles, so its gate is connected to what is now the drain. In this condition the device resembles a diode with the anode connected to VSS.

There is a parasitic pn junction between the ohmic resistor and the substrate. In the ROM parts (8051, 8048, etc.) the substrate is held at approximately  $-3V$  by the on-chip back-bias generator. In the EPROM parts (8751, 8748, etc.) the substrate is connected to VSS.

The effect of the input protection circuitry on the oscillator is that if the XTAL1 signal goes negative, its negative peak is clamped to  $-V_{DS}$  of the protection FET in the ROM parts, and to about  $-0.5V$  in the EPROM parts. These negative voltages on XTAL1 are in this application self-limiting and nondestructive.

The clamping action does, however, raise the DC level at XTAL1, which in turn tends to reduce the positive peak at XTAL2. The waveform at XTAL2 resembles a sinusoid riding on a DC level, and whose negative peaks are clipped off at zero.

Since it's normally the XTAL2 signal that drives the internal clocking circuitry, the question naturally arises as to how large this signal must be to reliably do its job. In fact, the XTAL2 signal doesn't have to meet the same VIH and VIL specifications that an external driver would have to. That's because as long as the oscillator is working, the on-chip amplifier is driving itself through its own 0-to-1 transition region, which is very nearly the same as the 0-to-1 transition region in the internal buffer that follows the oscillator. If some processing variations move the transition level higher or lower, the on-chip amplifier tends to compensate for it by the fact that its own transition level is correspondingly higher or lower. (In the 8096, it's the XTAL1 signal that drives the internal clocking circuitry, but the same concept applies.)

The main concern about the XTAL2 signal amplitude is as an indication of the general health of the oscillator. An amplitude of less than about 2.5V peak-to-peak indicates that start-up problems could develop in some units (with low gain) with some crystals (with high  $R_1$ ). The remedy is to either adjust the values of  $C_{X1}$  and/or  $C_{X2}$  or use a crystal with a lower  $R_1$ .

The amplitudes at XTAL1 and XTAL2 can be adjusted by changing the ratio of the capacitors from XTAL1 and XTAL2 to ground. Increasing the XTAL2 capacitance, for example, decreases the amplitude at XTAL2 and increases the amplitude at XTAL1 by about the same amount. Decreasing both caps increases both amplitudes.

### Pin Capacitance

Internal pin-to-ground and pin-to-pin capacitances at XTAL1 and XTAL2 will have some effect on the oscillator. These capacitances are normally taken to be in the range of 5 to 10 pF, but they are extremely difficult to evaluate. Any measurement of one such capacitance will necessarily include effects from the others. One advantage of the positive reactance oscillator is that the pin-to-ground capacitances are paralleled by external bulk capacitors, so a precise determination of their value is unnecessary. We would suggest that there is little justification for more precision than to assign them a value of 7 pF (XTAL1-to-ground and XTAL1-to-XTAL2). This value is probably not in error by more than 3 or 4 pF.

The XTAL2-to-ground capacitance is not entirely "pin capacitance," but more like an "equivalent output capacitance" of some 25 to 30 pF, having to include the effect of internal phase delays. This value will vary to some extent with temperature, processing, and frequency.

### MCS<sup>®</sup>-51 Oscillator

The on-chip amplifier on the HMOS MCS-51 family is shown in Figure 20. The drain load and feedback "resistors" are seen to be field-effect transistors. The drain load FET,  $R_D$ , is typically equivalent to about 1k to 3k-ohms. As an amplifier, the low frequency voltage gain is normally between -10 and -20, and the output resistance is effectively  $R_D$ .

The 8051 oscillator is normally used with equal bulk capacitors placed externally from XTAL1 to ground and from XTAL2 to ground. To determine a reasonable value of capacitance to use in these positions, given a crystal or ceramic resonator of known parameters, one can use the BASIC analysis in Appendix II to generate curves such as in Figures 17 and 18. This procedure will define a range of values that will minimize start-up time. We don't suggest that smaller values be used than those which minimize start-up time. Larger values than those can be used in applications where increased frequency stability is desired, at some sacrifice in start-up time.

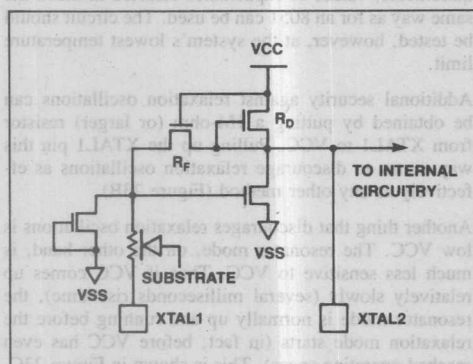


Figure 20 — MCS<sup>®</sup>-51 Oscillator Amplifier

Standard Crystal Corp. (reference 8) studied the use of their crystals with the MCS-51 family using skew samples supplied by Intel. They suggest putting 30 pF capacitors from XTAL1 and XTAL2 to ground, if the crystal is specified as described in reference 8. They noted that in that configuration and with crystals thus specified, the frequency accuracy was  $\pm 0.01\%$  and the frequency stability was  $\pm 0.005\%$ , and that a frequency accuracy of  $\pm 0.005\%$  could be obtained by substituting a 25 pF fixed cap in parallel with a 5-20 pF trimmer for one of the 30 pF caps.

MCS-51 skew samples have also been supplied to a number of ceramic resonator manufacturers for characterization with their products. These companies should be contacted for application information on their prod-

ucts. In general, however, ceramics tend to want somewhat larger values for  $C_{X1}$  and  $C_{X2}$  than quartz crystals do. As shown in Figure 18, they start up a lot faster that way.

In some applications the actual frequency tolerance required is only 1% or so, the user being concerned mainly that the circuit will oscillate. In that case,  $C_{X1}$  and  $C_{X2}$  can be selected rather freely in the range of 20 to 80 pF.

As you can see, "best" values for these components and their tolerances are strongly dependent on the application and its requirements. In any case, their suitability should be verified by environmental testing before the design is submitted to production.

### MCS<sup>®</sup>-48 Oscillator

The NMOS and HMOS MCS-48 oscillator is shown in Figure 21. It differs from the 8051 in that its inverting amplifier is a Schmitt Trigger. This configuration was

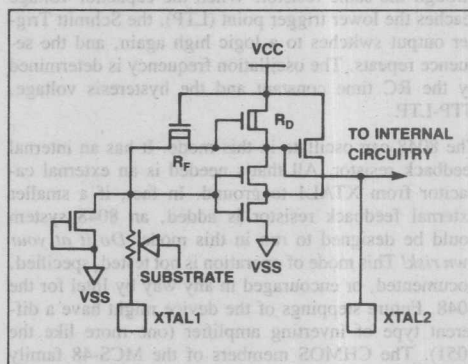
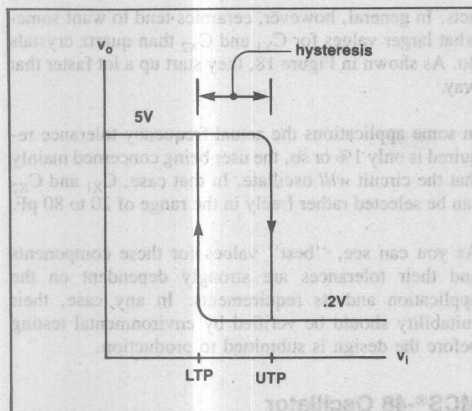


Figure 21 — MCS<sup>®</sup>-48 Oscillator Amplifier

chosen to prevent crosstalk from the TO pin, which is adjacent to the XTAL1 pin.

All Schmitt Trigger circuits exhibit a hysteresis effect, as shown in Figure 22. The hysteresis is what makes it less sensitive to noise. The same hysteresis allows any Schmitt Trigger to be used as a relaxation oscillator. All you have to do is connect a resistor from output to input, and a capacitor from input to ground, and the circuit oscillates in a relaxation mode as follows.

If the Schmitt Trigger output is at a logic high, the capacitor commences charging through the feedback resistor. When the capacitor voltage reaches the upper trigger point (UTP), the Schmitt Trigger output switches to a logic low and the capacitor commences discharging



**Figure 22 — Schmitt Trigger Characteristic**

through the same resistor. When the capacitor voltage reaches the lower trigger point (LTP), the Schmitt Trigger output switches to a logic high again, and the sequence repeats. The oscillation frequency is determined by the RC time constant and the hysteresis voltage, UTP-LTP.

The 8048 can oscillate in this mode. It has an internal feedback resistor. All that's needed is an external capacitor from XTAL1 to ground. In fact, if a smaller external feedback resistor is added, an 8048 system could be designed to run in this mode. *Do it at your own risk!* This mode of operation is not tested, specified, documented, or encouraged in any way by Intel for the 8048. Future steppings of the device might have a different type of inverting amplifier (one more like the 8051). The CHMOS members of the MCS-48 family do not use a Schmitt Trigger as the inverting amplifier.

Relaxation oscillations in the 8048 must be avoided, and this is the major objective in selecting the off-chip components needed to complete the oscillator circuit.

When an 8048 is powered up, if VCC has a short rise time, the relaxation mode starts first. The frequency is normally about 50kHz. The resonator mode builds more slowly, but it eventually takes over and dominates the operation of the circuit. This is shown in Figure 23A.

Due to processing variations, some units seem to have a harder time coming out of the relaxation mode, particularly at low temperatures. In some cases the resonator oscillations may fail entirely, and leave the device in the relaxation mode. Most units will stick in the relaxation mode at any temperature if  $C_{X1}$  is larger than about 50 pF. Therefore,  $C_{X1}$  should be chosen with some care, particularly if the system must operate at lower temperatures.

One method that has proven effective in all units to  $-40^{\circ}\text{C}$  is to put 5 pF from XTAL1 to ground and 20 pF from XTAL2 to ground. Unfortunately, while this method does discourage the relaxation mode, it is not an optimal choice for the resonator mode. For one thing, it does not swamp the pin capacitance. Also, it makes for a rather high signal level at XTAL1 (8 or 9 volts peak-to-peak).

The question arises as to whether that level of signal at XTAL1 might damage the chip. Not to worry. The negative peaks are self-limiting and nondestructive. The positive peaks could conceivably damage the oxide, but in fact, NMOS chips (eg, 8048) and HMOS chips (eg, 8048H) are tested to a much higher voltage than that. The technology trend, of course, is to thinner oxides, as the devices shrink in size. For an extra margin of safety, the HMOS II chips (eg, 8048AH) have an internal diode clamp at XTAL1 to VCC.

In reality,  $C_{X1}$  doesn't have to be quite so small to avoid relaxation oscillations, if the minimum operating temperature is not  $-40^{\circ}\text{C}$ . For less severe temperature requirements, values of capacitance selected in much the same way as for an 8051 can be used. The circuit should be tested, however, at the system's lowest temperature limit.

Additional security against relaxation oscillations can be obtained by putting a 1M-ohm (or larger) resistor from XTAL1 to VCC. Pulling up the XTAL1 pin this way seems to discourage relaxation oscillations as effectively as any other method (Figure 23B).

Another thing that discourages relaxation oscillations is low VCC. The resonator mode, on the other hand, is much less sensitive to VCC. Thus if VCC comes up relatively slowly (several milliseconds rise time), the resonator mode is normally up and running before the relaxation mode starts (in fact, before VCC has even reached operating specs). This is shown in Figure 23C.

A secondary effect of the hysteresis is a shift in the oscillation frequency. At low frequencies, the output signal from an inverter without hysteresis leads (or lags) the input by 180 degrees. The hysteresis in a Schmitt Trigger, however, causes the output to lead the input by less than 180 degrees (or lag by more than 180 degrees), by an amount that depends on the signal amplitude, as shown in Figure 24. At higher frequencies, there are additional phase shifts due to the various reactances in the circuit, but the phase shift due to the hysteresis is still present. Since the total phase shift in the oscillator's loop gain is necessarily 0 or 360 degrees, it is apparent that as the oscillations build up, the frequency has to change to allow the reactances to compensate for the hysteresis. In normal operation, this additional phase shift due to hysteresis does not exceed a few degrees, and the resulting frequency shift is negligible.

Kyocera, a ceramic resonator manufacturer, studied the

use of some of their resonators (at 6.0MHz, 8.0MHz, and 11.0MHz) with the 8049H. Their conclusion as to the value of capacitance to use at XTAL1 and XTAL2 was that 33 pF is appropriate at all three frequencies. One should probably follow the manufacturer's rec-

ommendations in this matter, since they will guarantee operation.

Whether one should accept these recommendations and guarantees without further testing is, however, another

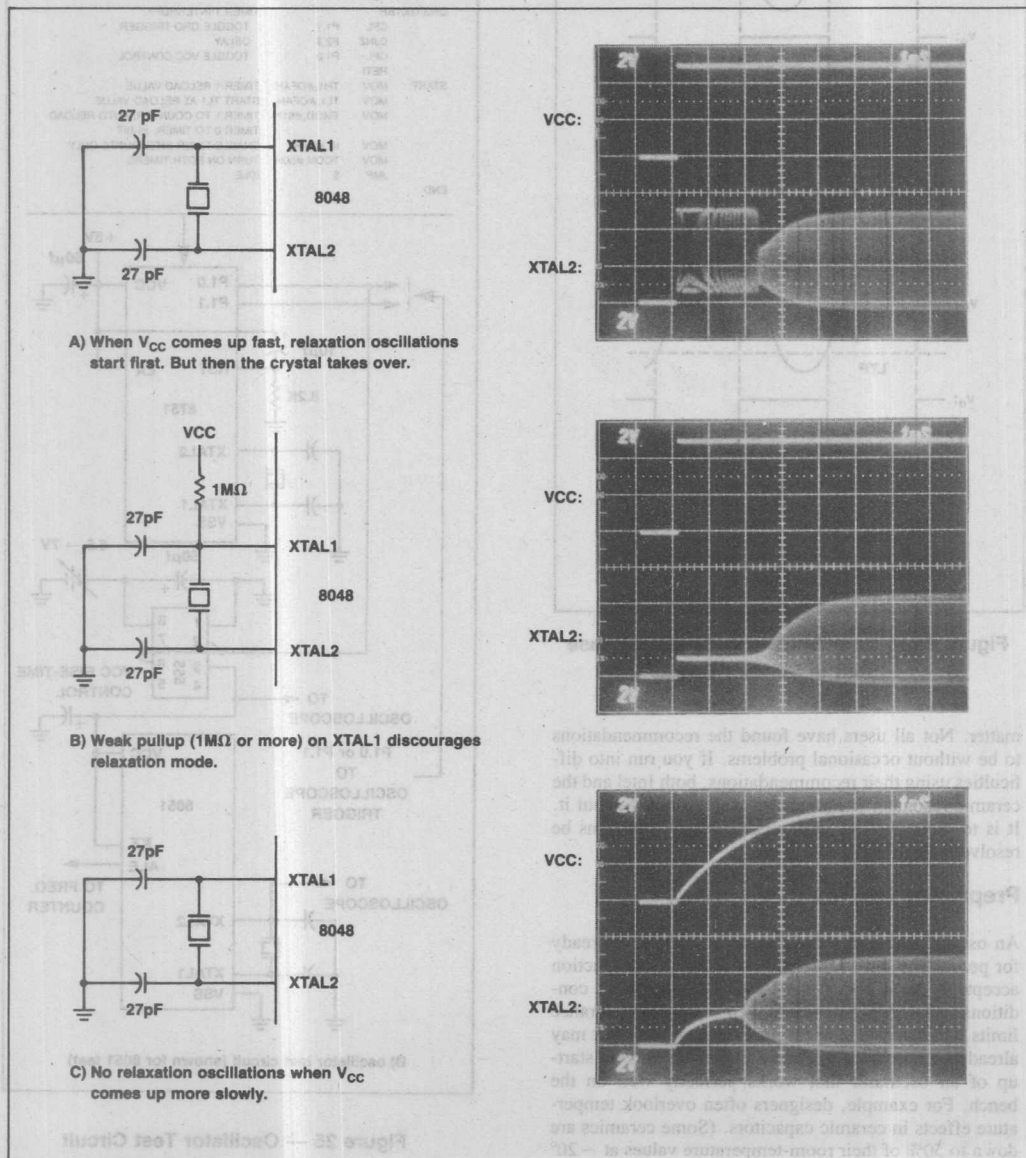


Figure 23 — Relaxation Oscillations in the 8048



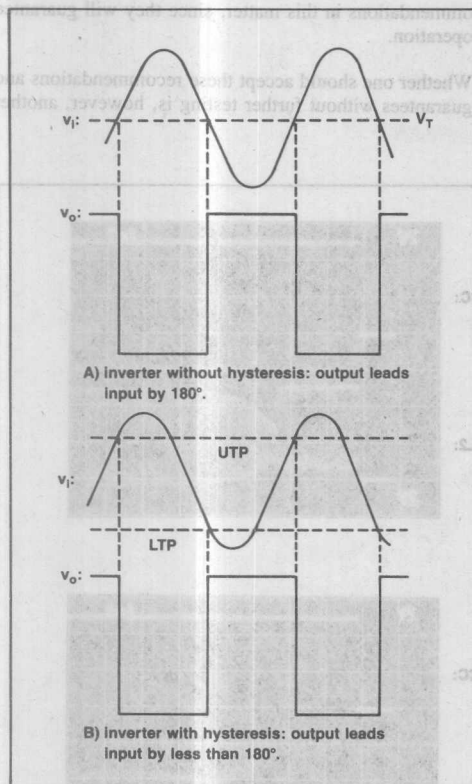


Figure 24 — Amplitude — Dependent Phase Shift in Schmitt Trigger

matter. Not all users have found the recommendations to be without occasional problems. If you run into difficulties using their recommendations, both Intel and the ceramic resonator manufacturer want to know about it. It is to their interest, and ours, that such problems be resolved.

### Preproduction Tests

An oscillator design should never be considered ready for production until it has proven its ability to function acceptably well under worst-case environmental conditions and with parameters at their worst-case tolerance limits. Unexpected temperature effects in parts that may already be near their tolerance limits can prevent start-up of an oscillator that works perfectly well on the bench. For example, designers often overlook temperature effects in ceramic capacitors. (Some ceramics are down to 50% of their room-temperature values at  $-20^{\circ}\text{C}$  and  $+60^{\circ}\text{C}$ .) The problem here isn't just one of

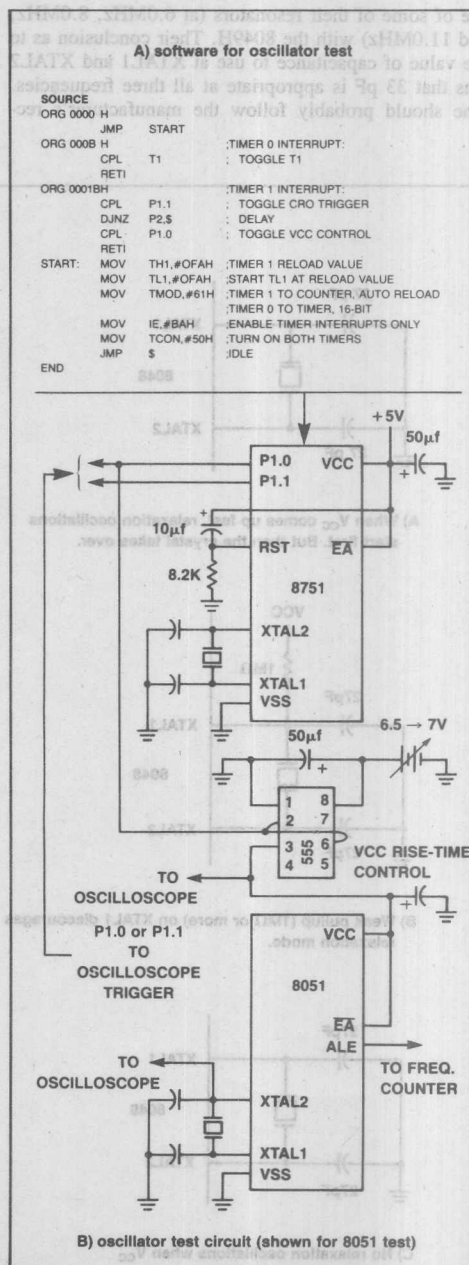


Figure 25 — Oscillator Test Circuit and Software

frequency stability, but also involves start-up time and steady-state amplitude. There may also be temperature effects in the resonator and amplifier.

It will be helpful to build a test jig that will allow the oscillator circuit to be tested independently of the rest of the system. Both start-up and steady-state characteristics should be tested. Figure 25 shows the circuit that was used to obtain the oscillator start-up photographs in this Application Note. This circuit or a modified version of it would make a convenient test vehicle. The oscillator and its relevant components can be physically separated from the control circuitry, and placed in a temperature chamber.

Start-up should be observed under a variety of conditions, including low VCC and using slow and fast VCC rise times. The oscillator should not be reluctant to start up even when VCC is below its spec value for the rest of the chip. (The rest of the chip may not function, but the oscillator should work.) It should also be verified that start-up occurs when the resonator has more than its upper tolerance limit of series resistance. (Put some resistance in series with the resonator for this test.) The bulk capacitors from XTAL1 and XTAL2 to ground should also be varied to their tolerance limits.

The same circuit, with appropriate changes in the software to lengthen the "on" time, can be used to test the steady-state characteristics of the oscillator, specifically the frequency, frequency stability, and amplitudes at XTAL1 and XTAL2.

As previously noted, the voltage swings at these pins are not critical, but they should be checked at the system's temperature limits to ensure that they are in good health. Observing these signals necessarily changes them somewhat. Observing the signal at XTAL2 requires that the capacitor at that pin be reduced to account for

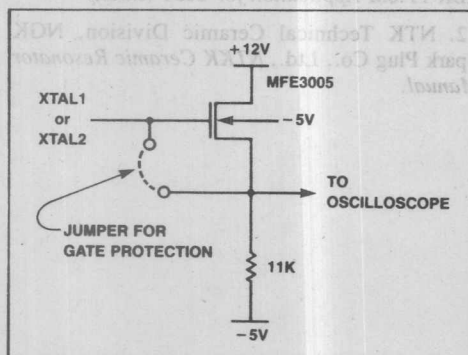


Figure 26 — MOSFET Buffer for Observing Oscillator Signals

the oscilloscope probe capacitance. Observing the signal at XTAL1 requires the same consideration, plus a blocking capacitor (switch the oscilloscope input to AC), so as to not disturb the DC level at that pin. Alternatively, a MOSFET buffer such as the one shown in Figure 26 can be used. It should be verified by direct measurement that the ground clip on the scope probe is ohmically connected to the scope chassis (probes are incredibly fragile in this respect), and the observations should be made with the ground clip on the VSS pin, or very close to it. If the probe shield isn't operational and in use, the observations are worthless.

Frequency checks should be made with only the oscillator circuitry connected to XTAL1 and XTAL2. The ALE frequency can be counted, and the oscillator frequency derived from that. In systems where the frequency tolerance is only "nominal," the frequency should still be checked to ascertain that the oscillator isn't running in a spurious resonance or relaxation mode. Switching VCC off and on again repeatedly will help reveal a tendency to go into unwanted modes of oscillation.

The operation of the oscillator should then be verified under actual system running conditions. By this stage one will be able to have some confidence that the basic selection of components for the oscillator itself is suitable, so if the oscillator appears to malfunction in the system the fault is not in the selection of these components.

### Troubleshooting Oscillator Problems

The first thing to consider in case of difficulty is that between the test jig and the actual application there may be significant differences in stray capacitances, particularly if the actual application is on a multi-layer board.

Noise glitches, that aren't present in the test jig but are in the application board, are another possibility. Capacitive coupling between the oscillator circuitry and other signals has already been mentioned as a source of miscounts in the internal clocking circuitry. Inductive coupling is also possible, if there are strong currents nearby. These problems are a function of the PCB layout.

Surrounding the oscillator components with "quiet" traces (VCC and ground, for example) will alleviate capacitive coupling to signals that have fast transition times. To minimize inductive coupling, the PCB layout should minimize the areas of the loops formed by the oscillator components. These are the loops that should be checked:

- XTAL1 through the resonator to XTAL2;
- XTAL1 through CX1 to the VSS pin;
- XTAL2 through CX2 to the VSS pin.

It is not unusual to find that the grounded ends of  $C_{X1}$  and  $C_{X2}$  eventually connect up to the VSS pin only after looping around the farthest ends of the board. Not good.

as to not disturb the DC level at that pin. Alternatively, a MOSFET buffer such as the one shown in Figure 26 can be used. It should be verified by direct measurement that the ground clip on the scope probe is electrically connected to the scope chassis (probes are incredibly fragile in this respect), and the connection should be made with the ground clip on the VSS pin, or very close to it. If the probe shield isn't operational and in use, the operations are worthless.

Frequency checks should be made with only the oscillator circuit connected to XTAL1 and XTAL2. The later circuitry can be counted, and the oscillator frequency derived from that. In systems where the frequency tolerance is only "nominal," the frequency should still be checked to ascertain that the oscillator isn't running in a spurious resonance or relaxation mode. Turning VCC off and on again repeatedly will help reveal a tendency to go into unwanted modes of operation.

1. Frerking, M. E., *Crystal Oscillator Design and Temperature Compensation*, Van Nostrand Reinhold, 1978.

2. Bottom, V., "The Crystal Unit as a Circuit Component," Ch. 7, *Introduction to Quartz Crystal Unit Design*, Van Nostrand Reinhold, 1982.

3. Parzen, B., *Design of Crystal and Other Harmonic Oscillators*, John Wiley & Sons, 1983.

4. Holmbeck, J. D., "Frequency Tolerance Limitations with Logic Gate Clock Oscillators," *31st Annual Frequency Control Symposium*, June, 1977.

5. Roberge, J. K., "Nonlinear Systems," Ch. 6, *Operational Amplifiers: Theory and Practice*, Wiley, 1975.

6. Eaton, S. S., *Timekeeping Advances Through COS/MOS Technology*, RCA Application Note ICAN-6086.

These problems are a function of the PCB layout. It is also possible, if there are strong currents nearby, that the oscillator component will "drift."

Summarizing the oscillator components with "drift," traces (VCC and ground, for example) will alternate capacitive coupling to signals that have fast transition times. To minimize inductive coupling, the PCB layout should minimize the areas of the loops formed by the oscillator components. These are the loops that should be checked.

XTAL1 through the resonator to XTAL2.  
XTAL1 through CX1 to the VSS pin.  
XTAL2 through CX2 to the VSS pin.

Finally, it should not be overlooked that software problems sometimes imitate the symptoms of a slow-starting oscillator or incorrect frequency. Never underestimate the perversity of a software problem.

It will be helpful to build a test jig that will allow the oscillator circuit to be tested independently of the rest of the system. Both start-up and steady-state characteristics should be tested. Figure 25 shows the circuit that was used to obtain the oscillator start-up photographs in this Application Note. This circuit or a modified version of it would make a convenient test vehicle. The oscillator and its relevant components can be physically separated from the control circuitry, and placed in a temperature chamber.

Start-up should be observed under a variety of conditions, including low VCC and rising slow and fast VCC times. The oscillator should not be reluctant to start up even when VCC is below its spec value for the test. (The test of the chip may not function, but the oscillator should work.) It should also be verified that start-up occurs when the resonator has more than its normal tolerance limit of 2 parts per million.

## REFERENCES

7. Eaton, S. S., *Micropower Crystal-Controlled Oscillator Design Using RCA COS/MOS Inverters*, RCA Application Note, ICAN-6539.

8. Fisher, J. B., *Crystal Specifications for the Intel 8031/8051/8751 Microcontrollers*, Standard Crystal Corp. Design Data Note #2F.

9. Murata Mfg. Co., Ltd., *Ceramic Resonator "Ceralock" Application Manual*.

10. Kyoto Ceramic Co., Ltd., *Adaptability Test Between Intel 8049H and Kyocera Ceramic Resonators*.

11. Kyoto Ceramic Co., Ltd., *Technical Data on Ceramic Resonator Model KBR-6.0M, KBR-8.0M, KBR-11.0M Application for 8051 (Intel)*.

12. NTK Technical Ceramic Division, NGK Spark Plug Co., Ltd., *NTKK Ceramic Resonator Manual*.

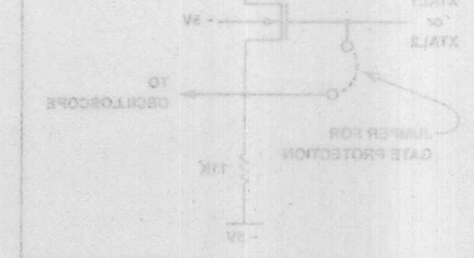


Figure 26 — MOSFET Buffer for Oscillator Signals

# APPENDIX I QUARTZ AND CERAMIC RESONATOR FORMULAS

Based on the equivalent circuit of the crystal, the impedance of the crystal is

$$Z_{XTAL} = \frac{(R_1 + j\omega L_1 + 1/j\omega C_1)(j\omega C_0)}{R_1 + j\omega L_1 + 1/j\omega C_1 + j\omega C_0}$$

After some algebraic manipulation, this calculation can be written in the form

$$Z_{XTAL} = \frac{1}{j\omega C_1 + C_0} \frac{1 - \omega^2 L_1 C_1 + j\omega R_1 C_1}{1 - \omega^2 L_1 C_1 + j\omega R_1 C_1}$$

where  $C_T$  is the capacitance of  $C_1$  in series with  $C_0$

$$C_T = \frac{C_1 C_0}{C_1 + C_0}$$

The impedance of the crystal in parallel with an external load capacitance  $C_L$  is the same expression, but with  $C_0 + C_L$  substituted for  $C_0$

$$Z_{XTAL||CL} = \frac{1}{j\omega(C_1 + C_0 + C_L)} \frac{1 - \omega^2 L_1 C_1 + j\omega R_1 C_1}{1 - \omega^2 L_1 C_1 + j\omega R_1 C_1}$$

where  $C_T$  is the capacitance of a series with  $(C_0 + C_L)$

$$C_T = \frac{C_1(C_0 + C_L)}{C_1 + C_0 + C_L}$$

The impedance of the crystal in series with the load capacitance is

$$Z_{XTAL+CL} = \frac{1}{j\omega C_1} \frac{1 - \omega^2 L_1 C_1 + j\omega R_1 C_1}{1 - \omega^2 L_1 C_1 + j\omega R_1 C_1 + j\omega C_0}$$

where  $C_T$  and  $C_L$  are as defined above.

The phase angles of these impedances are readily obtained from the impedance expressions themselves:

$$\theta_{XTAL} = \arctan \frac{\omega R_1 C_1}{1 - \omega^2 L_1 C_1}$$

$$\theta_{XTAL+CL} = \arctan \frac{\omega R_1 C_1}{1 - \omega^2 L_1 C_1 - \omega^2 L_1 C_0}$$

$$\theta_{XTAL||CL} = \arctan \frac{\omega R_1 C_1}{1 - \omega^2 L_1 C_1}$$

$$\theta_{XTAL||CL} = \arctan \frac{\omega R_1 C_1}{1 - \omega^2 L_1 C_1}$$

$$\theta_{XTAL} = \arctan \frac{\omega R_1 C_1}{1 - \omega^2 L_1 C_1}$$

$$\theta_{XTAL+CL} = \arctan \frac{\omega R_1 C_1}{1 - \omega^2 L_1 C_1 - \omega^2 L_1 C_0}$$

The resonant ("series resonant") frequency is the frequency at which the phase angle is zero and the impedance is low. The antiresonant ("parallel resonant") frequency is the frequency at which the phase angle is zero and the impedance is high.

Each of the above 6 expressions contains two arctan functions. Setting the denominator of the argument of the first arctan function to zero gives (approximately) the "series resonant" frequency for that configuration. Setting the denominator of the argument of the second arctan function to zero gives (approximately) the "parallel resonant" frequency for that configuration.

For example, the resonant frequency of the crystal is the

## APPENDIX I ..... 22 APPENDIX II ..... 24

$$f_s = \frac{1}{2\pi\sqrt{L_1 C_1}}$$

It will be noted that the series resonant frequency of the "XTAL + CL" configuration (crystal in series with  $C_L$ ) is the same as the parallel resonant frequency of the "XTAL || CL" configuration (crystal in parallel with  $C_L$ ). This is the frequency at which

$$1 - \omega^2 L_1 C_1 = 0$$

$$\omega = \frac{1}{\sqrt{L_1 C_1}}$$

$$f_s = \frac{1}{2\pi\sqrt{L_1 C_1}}$$

This fact is used by crystal manufacturers in the process of calibrating a crystal to a specified load capacitance.

By subtracting the resonant frequency of the crystal from its antiresonant frequency, one can calculate the range of frequencies over which the crystal reactance is positive:

$$f_a - f_s = f \left( \sqrt{1 + C_0/C_1} - 1 \right)$$



## APPENDIX I

### QUARTZ AND CERAMIC RESONATOR FORMULAS

Based on the equivalent circuit of the crystal, the impedance of the crystal is

$$Z_{XTAL} = \frac{(R_1 + j\omega L_1 + 1/j\omega C_1)(1/j\omega C_0)}{R_1 + j\omega L_1 + 1/j\omega C_1 + 1/j\omega C_0}$$

After some algebraic manipulation, this calculation can be written in the form

$$Z_{XTAL} = \frac{1}{j\omega(C_1 + C_0)} \cdot \frac{1 - \omega^2 L_1 C_1 + j\omega R_1 C_1}{1 - \omega^2 L_1 C_T + j\omega R_1 C_T}$$

where  $C_T$  is the capacitance of  $C_1$  in series with  $C_0$ :

$$C_T = \frac{C_1 C_0}{C_1 + C_0}$$

The impedance of the crystal in parallel with an external load capacitance  $C_L$  is the same expression, but with  $C_0 + C_L$  substituted for  $C_0$ :

$$Z_{XTAL \parallel CL} = \frac{1}{j\omega(C_1 + C_0 + C_L)} \cdot \frac{1 - \omega^2 L_1 C_1 + j\omega R_1 C_1}{1 - \omega^2 L_1 C_T + j\omega R_1 C_T}$$

where  $C_T$  is the capacitance of  $C_1$  in series with  $(C_0 + C_L)$ :

$$C_T = \frac{C_1(C_0 + C_L)}{C_1 + C_0 + C_L}$$

The impedance of the crystal in series with the load capacitance is

$$\begin{aligned} Z_{XTAL + CL} &= Z_{XTAL} + \frac{1}{j\omega C_L} \\ &= \frac{C_L + C_1 + C_0}{j\omega C_L (C_1 + C_0)} \cdot \frac{1 - \omega^2 L_1 C_T + j\omega R_1 C_T}{1 - \omega^2 L_1 C_T + j\omega R_1 C_T} \end{aligned}$$

where  $C_T$  and  $C_T$  are as defined above.

The phase angles of these impedances are readily obtained from the impedance expressions themselves:

$$\begin{aligned} \theta_{XTAL} &= \arctan \frac{\omega R_1 C_1}{1 - \omega^2 L_1 C_1} \\ &\quad - \arctan \frac{\omega R_1 C_T}{1 - \omega^2 L_1 C_T} - \frac{\pi}{2} \\ \theta_{XTAL \parallel CL} &= \arctan \frac{\omega R_1 C_1}{1 - \omega^2 L_1 C_1} \\ &\quad - \arctan \frac{\omega R_1 C_T}{1 - \omega^2 L_1 C_T} - \frac{\pi}{2} \end{aligned}$$

$$\begin{aligned} \theta_{XTAL + CL} &= \arctan \frac{\omega R_1 C_T}{1 - \omega^2 L_1 C_T} \\ &\quad - \arctan \frac{\omega R_1 C_T}{1 - \omega^2 L_1 C_T} - \frac{\pi}{2} \end{aligned}$$

The resonant ("series resonant") frequency is the frequency at which the phase angle is zero and the impedance is low. The antiresonant ("parallel resonant") frequency is the frequency at which the phase angle is zero and the impedance is high.

Each of the above  $\theta$ -expressions contains two arctan functions. Setting the denominator of the argument of the first arctan function to zero gives (approximately) the "series resonant" frequency for that configuration. Setting the denominator of the argument of the second arctan function to zero gives (approximately) the "parallel resonant" frequency for that configuration.

For example, the resonant frequency of the crystal is the frequency at which

$$1 - \omega^2 L_1 C_1 = 0$$

$$\text{Thus, } \omega_s = \frac{1}{\sqrt{L_1 C_1}}$$

$$\text{or } f_s = \frac{1}{2\pi \sqrt{L_1 C_1}}$$

It will be noted that the series resonant frequency of the "XTAL + CL" configuration (crystal in series with  $C_L$ ) is the same as the parallel resonant frequency of the "XTAL  $\parallel$  CL" configuration (crystal in parallel with  $C_L$ ). This is the frequency at which

$$1 - \omega^2 L_1 C_T = 0$$

$$\text{Thus, } \omega_a = \frac{1}{\sqrt{L_1 C_T}}$$

$$\text{or } f_a = \frac{1}{2\pi \sqrt{L_1 C_T}}$$

This fact is used by crystal manufacturers in the process of calibrating a crystal to a specified load capacitance.

By subtracting the resonant frequency of the crystal from its antiresonant frequency, one can calculate the range of frequencies over which the crystal reactance is positive:

$$\begin{aligned} f_a - f_s &= f_s (\sqrt{1 + C_1/C_0} - 1) \\ &= f_s \left( \frac{C_1}{2C_0} \right) \end{aligned}$$

Given typical values for  $C_1$  and  $C_0$ , this range can hardly exceed 0.5% of  $f_s$ . Unless the inverting amplifier in the positive reactance oscillator is doing something very strange indeed, the oscillation frequency is bound to be accurate to that percentage whether the crystal was calibrated for series operation or to any unspecified load capacitance.

### Equivalent Series Resistance

ESR is the real part of  $Z_{XTAL}$  at the oscillation frequency. The oscillation frequency is the parallel resonant frequency of the "XTAL || CL" configuration (which is the same as the series resonant frequency of the "XTAL + CL" configuration). Substituting this frequency into the  $Z_{XTAL}$  expression yields, after some algebraic manipulation,

$$ESR = R_1 \left( \frac{C_0 + C_L}{C_L} \right)^2 \left( 1 + \omega^2 C_1^2 \left( \frac{C_0 + C_L}{C_L} \right)^2 \right)^{-1/2}$$

$$\approx R_1 \left( 1 + \frac{C_0^2}{C_L^2} \right)$$

Then the program asks for the values of the XTAL1-to-ground and XTAL1-to-XTAL2 capacitances. For XTAL1, enter the value of the externally connected bulk capacitor plus an estimated 7 pf for pin capacitance. For XTAL2, enter the value of the externally connected bulk capacitor plus an estimated 7 pf for pin capacitance plus about 50 pf to simulate high-frequency roll-off and phase shift in the on-chip circuitry.

Next the program asks for values for the small-signal parameters of the on-chip amplifier. Typically, for the

Amplifier Gain Magnitude = 12  
Feedback Resistance = 2300 ohms  
Output Resistance = 25 ohms

The same values can be used for MCS-48 (NMOS) and HMOS devices, but they are difficult to verify, because the Schmitt Trigger does not lend itself to small-signal measurements.

### Drive Level

The power dissipated by the crystal is  $I_1^2 R_1$ , where  $I_1$  is the RMS current in the motional arm of the crystal. This current is given by  $V_x / |Z_1|$ , where  $V_x$  is the RMS voltage across the crystal, and  $|Z_1|$  is the magnitude of the impedance of the motional arm. At the oscillation frequency, the motional arm is a positive (inductive) reactance in parallel resonance with  $(C_0 + C_L)$ . Therefore  $|Z_1|$  is approximately equal to the magnitude of the reactance of  $(C_0 + C_L)$ :

$$|Z_1| = \frac{1}{2\pi f(C_0 + C_L)}$$

where  $f$  is the oscillation frequency. Then,

$$P = I_1^2 R_1 = \left( \frac{V_x}{|Z_1|} \right)^2 R_1$$

$$= [2\pi f(C_0 + C_L) V_x]^2 R_1$$

The waveform of the voltage across the crystal (XTAL1 to XTAL2) is approximately sinusoidal. If its peak value is  $V_{CC}$ , then  $V_x$  is  $V_{CC}/\sqrt{2}$ . Therefore,

$$P = 2R_1[\pi f(C_0 + C_L)V_{CC}]^2$$

The MCS-48 family is specifically included in the program only to the extent that the input-output curve used in the steady-state analysis is that of a Schmitt Trigger. If the user identifies the device under analysis as an MCS-48 device, the analysis does not include the voltage-dependent phase shift of the Schmitt Trigger.

The clamping action of the input protection circuitry is important in determining the steady-state amplitudes. The steady-state routine accounts for it by setting the negative peak of the XTAL1 signal at a level which depends on the amplitude of the XTAL1 signal in accordance with experimental observations. It's an exercise in curve-fitting. A user may find a different type of curve works better. Later editions of the chips may behave differently in this respect, having somewhat different types of input protection circuitry.

## APPENDIX II

# OSCILLATOR ANALYSIS PROGRAM

The program is written in BASIC. BASIC is excruciatingly slow, but it has some advantages. For one thing, more people know BASIC than FORTRAN. In addition, a BASIC program is easy to develop, modify, and "fiddle around" with. Another important advantage is that a BASIC program can run on practically any small computer system.

Its slowness is a problem, however. For example, the routine which calculates the "start-up time constant" discussed in the text may take several hours to complete. A person who finds this program useful may prefer to convert it to FORTRAN, if the facilities are available.

### Limitations of the Program

The program was developed with specific reference to 8051-type oscillator circuitry. That means the on-chip amplifier is a simple inverter, and not a Schmitt Trigger. The 8096, the 80C51, the 80C48 and 80C49 all have simple inverters. The 8096 oscillator is almost identical to the 8051, differing mainly in the input protection circuitry. The CMOS amplifiers have somewhat different parameters (higher gain, for example), and different transition levels than the 8051.

The MCS-48 family is specifically included in the program only to the extent that the input-output curve used in the steady-state analysis is that of a Schmitt Trigger, if the user identifies the device under analysis as an MCS-48 device. The analysis does not include the voltage dependent phase shift of the Schmitt Trigger.

The clamping action of the input protection circuitry is important in determining the steady-state amplitudes. The steady-state routine accounts for it by setting the negative peak of the XTAL1 signal at a level which depends on the amplitude of the XTAL1 signal in accordance with experimental observations. It's an exercise in curve-fitting. A user may find a different type of curve works better. Later steppings of the chips may behave differently in this respect, having somewhat different types of input protection circuitry.

It should be noted that the analysis ignores a number of important items, such as high-frequency effects in the on-chip circuitry. These effects are difficult to predict, and are no doubt dependent on temperature, frequency, and device sample. However, they can be simulated to a reasonable degree by adding an "output capacitance" of about 20 pF to the circuit model (ie, in parallel with CX2), as described below.

### Notes on Using the Program

The program asks the user to input values for various circuit parameters. First the crystal (or ceramic resonator) parameters are asked for. These are R1, L1, C1, and C0. The manufacturer can supply these values for selected samples. To obtain any kind of correlation between calculation and experiment, the values of these parameters must be known for the specific sample in the test circuit. The value that should be entered for C0 is the C0 of the crystal itself plus an estimated 7 pF to account for the XTAL1-to-XTAL2 pin capacitance, plus any other stray capacitance paralleling the crystal that the user may feel is significant enough to be included.

Then the program asks for the values of the XTAL1-to-ground and XTAL2-to-ground capacitances. For CXTAL1, enter the value of the externally connected bulk capacitor plus an estimated 7 pF for pin capacitance. For CXTAL2, enter the value of the externally connected bulk capacitor plus an estimated 7 pF for pin capacitance plus about 20 pF to simulate high-frequency roll-off and phase shifts in the on-chip circuitry.

Next the program asks for values for the small-signal parameters of the on-chip amplifier. Typically, for the 8051/8751,

Amplifier Gain Magnitude	=	15
Feedback Resistance	=	2300k-ohms
Output Resistance	=	2k-ohms

The same values can be used for MCS-48 (NMOS and HMOS) devices, but they are difficult to verify, because the Schmitt Trigger does not lend itself to small-signal measurements.

```

100 DEFDBL C,D,F,G,L,P,R,S,X
200 REM
300 REM *****
400 REM
500 REM
600 REM
700 REM
800 REM FNZM(R,X) = MAGNITUDE OF A COMPLEX NUMBER, (R+JX)
900 DEF FNZM(R,X) = SQR(R^2+X^2)
1000 REM
1100 REM FNZP(R,X) = ANGLE OF A COMPLEX NUMBER
1200 REM      = 180/PI*ARCTAN(X/R) IF R>0
1300 REM      = 180/PI*ARCTAN(X/R) + 180 IF R<0 AND X>0
1400 REM      = 180/PI*ARCTAN(X/R) - 180 IF R<0 AND X<0
1500 DEF FNZP(R,X) = 180/PI*ATN(X/R) - (SGN(R)-1)*SGN(X)*90
1600 REM
1700 REM INDUCTIVE IMPEDANCE AT COMPLEX FREQUENCY S+JF (HZ)
1800 REM      Z = 2*PI*S*L + J*2*PI*F*L
1900 REM      = FNRL(S,L) + JFNXL(F,L)
2000 DEF FNRL(S,L) = 2*PI*S*L
2100 DEF FNXL(F,L) = 2*PI*F*L
2200 REM
2300 REM CAPACITIVE IMPEDANCE AT COMPLEX FREQUENCY S+JF (HZ)
2400 REM      Z = 1/[2*PI*(S+JF)*C]
2500 REM      = S/[2*PI*(S^2+F^2)*C] + J*(-F)/[2*PI*(S^2+F^2)*C]
2600 REM      = FNRC(S,F,C) + JFNXC(S,F,C)
2700 DEF FNRC(S,F,C) = SC/(2*PI*(S^2+F^2)*CC)
2800 DEF FNXC(S,F,C) = -FC/(2*PI*(S^2+F^2)*CC)
2900 REM
3000 REM RATIO OF TWO COMPLEX NUMBERS
3100 REM      RA+JXA RA*RB+XA*XB XA*RB-RA*XB
3200 REM      ----- = ----- + J -----
3300 REM      RB+JXB RB^2+XB^2 RB^2+XB^2
3400 REM      = FNRR(RA,XA,RB,XB) + JFNXR(RA,XA,RB,XB)
3500 DEF FNRR(RA,XA,RB,XB) = (RA*RB+XA*XB)/(RB^2+XB^2)
3600 DEF FNXR(RA,XA,RB,XB) = (XA*RB-RA*XB)/(RB^2+XB^2)
3700 REM
3800 REM PRODUCT OF TWO COMPLEX NUMBERS
3900 REM      (RA+JXA)*(RB+JXB) = RA*RB-XA*XB + J(XA*RB+RA*XB)
4000 REM      = FNRM(RA,XA,RB,XB) + JFNXM(RA,XA,RB,XB)
4100 DEF FNRM(RA,XA,RB,XB) = RA*RB - XA*XB
4200 DEF FNXM(RA,XA,RB,XB) = RA*XB + RB*XA
4300 REM
4400 REM
4500 REM PARALLEL IMPEDANCES
4600 REM      (RA+JXA):(RB+JXB) = -----
4700 REM      RA+RB + J(XA+XB)
4800 REM
4900 REM
5000 REM      RA*(RB^2+XB^2)+RB*(RA^2+XA^2) XA*(RB^2+XB^2)+XB*(RA^2+XA^2)
5100 REM      = ----- + J -----
5200 REM      (RA+RB)^2 + (XA+XB)^2 (RA+RB)^2 + (XA+XB)^2
5300 REM
5400 REM      = FNRP(RA,XA,RB,XB) + JFNXP(RA,XA,RB,XB)
5500 DEF FNRP(RA,XA,RB,XB) = (RA*(RB^2+XB^2) + RB*(RA^2+XA^2))/(RA+RB)^2 + (XA+XB)^2
5600 DEF FNXP(RA,XA,RB,XB) = (XA*(RB^2+XB^2) + XB*(RA^2+XA^2))/(RA+RB)^2 + (XA+XB)^2
5700 REM
5800 REM *****
5900 REM
6000 REM BEGIN COMPUTATIONS
6100 REM
6200 LET PI = 3.141592654#
6300 REM
6400 REM DEFINE CIRCUIT PARAMETERS
6500 GOSUB 14500
6600 REM
6700 REM ESTABLISH NOMINAL RESONANT AND ANTIRESONANT CRYSTAL FREQUENCIES
6800 FS = FIX(1/(2*PI*SQR(L1*C1)))
6900 FA = FIX(1/(2*PI*SQR(L1*C1*CO/(C1+CO))))
7000 PRINT
7100 PRINT "XTAL IS SERIES RESONANT AT ",FS," HZ"
7200 PRINT " PARALLEL RESONANT AT ",FA," HZ"
7300 PRINT
7400 PRINT "SELECT: 1. LIST PARAMETERS"
7500 PRINT " 2. CIRCUIT ANALYSIS"
7600 PRINT " 3. OSCILLATION FREQUENCY"
7700 PRINT " 4. START-UP TIME CONSTANT"
7800 PRINT " 5. STEADY-STATE ANALYSIS"

```

APRIL 8, 1983



```

7900 PRINT
8000 INPUT N
8100 IF N=1 THEN PRINT ELSE 8600
8200 REM
8300 REM ----- LIST PARAMETERS -----
8400 GOSUB 17100
8500 GOTO 6800
8600 IF N=2 THEN PRINT ELSE 9400
8700 REM
8800 REM ----- CIRCUIT ANALYSIS -----
8900 PRINT " FREQUENCY S+JF TYPE (S), (F)."
9000 INPUT SQ,FQ
9100 GOSUB 20200
9200 GOSUB 26600
9300 GOTO 6800
9400 IF N=3 THEN 10300 ELSE 11000
9500 REM
9600 REM ----- OSCILLATION FREQUENCY -----
9700 CL = CX*CY/(CX+CY) + CO
9800 FQ = FIX(1/(2*PI*SQR(L1*C1*CL/(C1+CL))))
9900 SQ = 0
10000 DF = FIX(10^INT(LOG(FA-FS)/LOG(10)-2)+.5)
10100 DS = 0
10200 RETURN
10300 GOSUB 9700
10400 GOSUB 30300
10500 PRINT
10600 PRINT
10700 PRINT "FREQUENCY AT WHICH LOOP GAIN HAS ZERO PHASE ANGLE:"
10800 GOSUB 26600
10900 GOTO 6800
11000 IF N=4 THEN PRINT ELSE 12200
11100 REM
11200 REM ----- START-UP TIME CONSTANT -----
11300 PRINT "THIS WILL TAKE SOME TIME"
11400 GOSUB 9700
11500 GOSUB 37700
11600 PRINT
11700 PRINT
11800 PRINT "FREQUENCY AT WHICH LOOP GAIN = 1 AT 0 DEGREES:"
11900 GOSUB 26600
12000 PRINT "PRINT THIS YIELDS A START-UP TIME CONSTANT OF";CSNG(1000000/(2*PI*SQ));"MICROSECS"
12100 GOTO 6800
12200 IF N=5 THEN PRINT ELSE 7300
12300 REM
12400 REM ----- STEADY-STATE ANALYSIS -----
12500 PRINT "STEADY-STATE ANALYSIS"
12600 PRINT
12700 PRINT "SELECT: 1. 8031/8051"
12800 PRINT "2. 8751"
12900 PRINT "3. 8035/8039/8040/8048/8049"
13000 PRINT "4. 8748/8749"
13100 INPUT ICX
13200 IF ICX<1 OR ICX>4 THEN 12600
13300 GOSUB 46900
13400 GOTO 7300
13500 REM SUBROUTINE BELOW DEFINES INPUT-OUTPUT CURVE OF OSCILLATOR CKT
13600 IF ICX>2 AND VO=5 AND VI<2 THEN RETURN
13700 VO = -10*VI + 15
13800 IF VO>5 THEN VO = 5
13900 IF VO<2 THEN VO = 2
14000 IF ICX>2 AND VO>2 THEN VO = 5
14100 RETURN
14200 REM
14300 REM *****
14400 REM
14500 REM DEFINE CIRCUIT PARAMETERS
14600 REM
14700 INPUT " R1 (OHMS)";R1
14800 INPUT " L1 (HENRY)";L1
14900 INPUT " C1 (PF)";X
15000 C1 = X*1E-12
15100 INPUT " CQ (PF)";X
15200 CO = X*1E-12
15300 INPUT " CXTAL1 (PF)";X
15400 CX = X*1E-12
15500 INPUT " CXTAL2 (PF)";X
15600 CY = X*1E-12

```

```

15700 INPUT " GAIN FACTOR MAGNITUDE";AV#
15800 INPUT " AMP FEEDBACK RESISTANCE (K-OHMS)";X
15900 RX = X*1000#
16000 INPUT " AMP OUTPUT RESISTANCE (K-OHMS)";X
16100 RO = X*1000#
16200 REM
16300 REM
16400 REM      LIST CURRENT PARAMETER VALUES
16500 GOSUB 17100
16600 RETURN
16700 REM
16800 REM
16900 REM *****
17000 REM
17100 REM      LIST CURRENT PARAMETER VALUES
17200 REM
17300 PRINT
17400 PRINT "CURRENT PARAMETER VALUES: 1. R1 = ";R1;" OHMS"
17500 PRINT "2. L1 = ";CSNG(L1);" HENRY"
17600 PRINT "3. C1 = ";CSNG(C1*1E+12);" PF"
17700 PRINT "4. CO = ";CSNG(CO*1E+12);" PF"
17800 PRINT "5. CXTAL1 = ";CSNG(CX*1E+12);" PF"
17900 PRINT "6. CXTAL2 = ";CSNG(CY*1E+12);" PF"
18000 PRINT "7. AMPLIFIER GAIN MAGNITUDE = ";AV#
18100 PRINT "8. FEEDBACK RESISTANCE = ";CSNG(RX*.001);" K-OHMS"
18200 PRINT "9. OUTPUT RESISTANCE = ";CSNG(RO*.001);" K-OHMS"
18300 PRINT
18400 PRINT "TO CHANGE A PARAMETER VALUE, TYPE (PARAM. NO.), (NEW VALUE)."
18500 PRINT "OTHERWISE, TYPE 0.0"
18600 INPUT NX,X
18700 IF NX=0 THEN RETURN
18800 IF NX=1 THEN R1 = X
18900 IF NX=2 THEN L1 = X
19000 IF NX=3 THEN C1 = X*1E-12
19100 IF NX=4 THEN CO = X*1E-12
19200 IF NX=5 THEN CX = X*1E-12
19300 IF NX=6 THEN CY = X*1E-12
19400 IF NX=7 THEN AV# = X
19500 IF NX=8 THEN RX = X*1000#
19600 IF NX=9 THEN RO = X*1000#
19700 GOTO 17400
19800 REM
19900 REM
20000 REM *****
20100 REM
20200 REM      CIRCUIT ANALYSIS
20300 REM
20400 REM      This routine calculates the loop gain at complex frequency SQ+JFQ.
20500 REM
20600 REM      1. Crystal impedance: RE + jXE
20700 REM
20800 X1 = FNXL(FG,L1) + FNXC(SG,FG,C1)
20900 RE = FNRP((R1+FNRL(SG,L1)+FNRC(SG,FG,C1)),X1,FNRC(SG,FG,CO),FNXC(SG,FG,CO))
21000 XE = FNXP((R1+FNRL(SG,L1)+FNRC(SG,FG,C1)),X1,FNRC(SG,FG,CO),FNXC(SG,FG,CO))
21100 REM
21200 REM      2. RF + jXF = (RE+jXE):(amplifier feedback resistance)
21300 REM
21400 RF = FNRP(RX,O,RE,XE)
21500 XF = FNXP(RX,O,RE,XE)
21600 REM
21700 REM      3. Input impedance: Zi = R1 + jXI = impedance of CXTAL1
21800 REM
21900 RI = FNRC(SG,FG,CX)
22000 XI = FNXC(SG,FG,CX)
22100 REM
22200 REM      4. Load impedance: ZL = (impedance of CXTAL2):(RF+RI)+(XF+XI)
22300 REM
22400 RL = FNRP((RF+RI),(XF+XI),FNRC(SG,FG,CY),FNXC(SG,FG,CY))
22500 XL = FNXP((RF+RI),(XF+XI),FNRC(SG,FG,CY),FNXC(SG,FG,CY))
22600 REM
22700 REM      5. Amplifier gain A = -AV*ZL/(ZL+RO)
22800 REM
22900 REM
23000 AR# = -AV*FNRR(RL,XL,(RO+RL),XL)
23100 AI# = -AV*FNXR(RL,XL,(RO+RL),XL)
23200 REM
23300 REM      6. Feedback ratio (beta) = (R1+jXI)/[(RF+RI)+(XF+XI)]
23400 REM

```

```

23500 REM
23600 BR# = FNRR(RI, XI, (RI+RF), (XI+XF))
23700 BI# = FNXR(RI, XI, (RI+RF), (XI+XF))
23800 REM
23900 REM 7. Amplifier gain in magnitude/phase form: AR+jAI = A at AP degrees
24000 REM
24100 A = FNZM(AR#, AI#)
24200 AP = FNZP(AR#, AI#)
24300 REM
24400 REM 8. (beta) in magnitude/phase form: BR+jBI = B at BP degrees
24500 REM
24600 B = FNZM(BR#, BI#)
24700 BP = FNZP(BR#, BI#)
24800 REM
24900 REM 9. Loop gain. G = (BR+jBI)*(AR+jAI)
25000 REM = G(real) + jG(imaginary)
25100 REM
25200 GR = FNRM(AR#, AI#, BR#, BI#)
25300 GI = FNXM(AR#, AI#, BR#, BI#)
25400 REM
25500 REM 10. Loop gain in magnitude/phase form: GR+jGI = AL at AQ degrees
25600 REM
25700 AL = FNZM(GR, GI)
25800 AQ = FNZP(GR, GI)
25900 RETURN
26000 REM
26100 REM
26200 REM *****
26300 REM
26400 REM PRINT CIRCUIT ANALYSIS RESULTS
26500 REM
26600 PRINT
26700 PRINT " FREQUENCY = "; SQ; " + J"; FG; " HZ"
26800 PRINT " XTAL IMPEDANCE = "; FNZM(RE, XE); " OHMS AT "; FNZP(RE, XE); " DEGREES"
26900 PRINT " (RE = "; CSNG(RE); " OHMS)"
27000 PRINT " (XE = "; CSNG(XE); " OHMS)"
27100 PRINT " LOAD IMPEDANCE = "; FNZM(RL, XL); " OHMS AT "; FNZP(RL, XL); " DEGREES"
27200 PRINT " AMPLIFIER GAIN = "; A; " AT "; AP; " DEGREES"
27300 PRINT " FEEDBACK RATIO = "; B; " AT "; BP; " DEGREES"
27400 PRINT " LOOP GAIN = "; AL; " AT "; AQ; " DEGREES"
27500 RETURN
27600 REM
27700 REM
27800 REM *****
27900 REM
28000 REM SEARCH FOR FREQUENCY (S+JF)
28100 REM AT WHICH LOOP GAIN HAS ZERO PHASE ANGLE
28200 REM
28300 REM This routine searches for the frequency at which the imaginary part
28400 REM of the loop gain is zero. The algorithm is as follows:
28500 REM 1. Calculate the sign of the imaginary part of the loop gain (GI).
28600 REM 2. Increment the frequency.
28700 REM 3. Calculate the sign of GI at the incremented frequency.
28800 REM 4. If the sign of GI has not changed, go back to 2.
28900 REM 5. If the sign of GI has changed, and this frequency is within
29000 REM 1Hz of the previous sign-change, exit the routine.
29100 REM 6. Otherwise, divide the frequency increment by -10.
29200 REM 7. Go back to 2.
29300 REM The routine is entered with the starting frequency SQ+jFG and
29400 REM starting increment DS+jDF already defined by the calling program.
29500 REM In actual use either DS or DF is zero, so the routine searches for
29600 REM a GI=0 point by incrementing either SQ or FG while holding the other
29700 REM constant. It returns control to the calling program with the
29800 REM incremented part of the frequency being within 1Hz of the actual
29900 REM GI=0 point.
30000 REM
30100 REM 1. CALCULATE THE SIGN OF THE IMAGINARY PART OF THE LOOP GAIN (GI).
30200 REM
30300 GOSUB 20200
30400 GOSUB 26600
30500 IF GI=0 THEN RETURN
30600 SX = INT(SGN(GI))
30700 IF SX=+1 THEN DS = -DS
30800 REM (REVERSAL OF DS FOR GI>0 IS FOR THE POLE-SEARCH ROUTINE.)
30900 REM
31000 REM 2. INCREMENT THE FREQUENCY.
31100 REM
31200 SP = SQ

```

```

31300 FP = FQ
31400 SQ = SQ + DS
31500 FQ = FQ + DF
31600 REM
31700 REM 3. CALCULATE THE SIGN OF GI AT THE INCREMENTED FREQUENCY.
31800 REM
31900 GOSUB 20200
32000 GOSUB 26600
32100 IF INT(SGN(GI))=0 THEN RETURN
32200 REM
32300 REM 4. IF THE SIGN OF GI HAS NOT CHANGED, GO BACK TO 2.
32400 REM
32500 IF SX%+INT(SGN(GI))=0 THEN PRINT ELSE 31400
32600 SX% = -SX%
32700 REM
32800 REM 5. IF THE SIGN OF GI HAS CHANGED, AND IF THIS FREQUENCY IS WITHIN
32900 REM 1HZ OF THE PREVIOUS SIGN-CHANGE, AND IF GI IS NEGATIVE, THEN
33000 REM EXIT THE ROUTINE (THE ADDITIONAL REQUIREMENT FOR NEGATIVE GI
33100 REM IS FOR THE POLE-SEARCH ROUTINE.)
33200 REM
33300 IF ABS(SP-SQ)<1 AND ABS(FP-FQ)<1 AND SX%=-1 THEN RETURN
33400 REM
33500 REM 6. DIVIDE THE FREQUENCY INCREMENT BY -10.
33600 REM
33700 DS = -DS/10#
33800 DF = -DF/10#
33900 REM
34000 REM 7. GO BACK TO 2.
34100 REM
34200 GOTO 31200
34300 REM
34400 REM
34500 REM *****
34600 REM
34700 REM SEARCH FOR POLE FREQUENCY
34800 REM
34900 REM This routine searches for the frequency at which the loop gain = 1
35000 REM at 0 degrees. That frequency is the pole frequency of the closed-
35100 REM loop gain function. The pole frequency is a complex number, SQ+jFQ
35200 REM (Hz). Oscillator start-up ensues if SQ>0. The algorithm is based on
35300 REM the calculated behavior of the phase angle of the loop gain in the
35400 REM region of interest on the complex plane. The locus of points of zero
35500 REM phase angle crosses the j-axis at the oscillation frequency and at
35600 REM some higher frequency. In between these two crossings of the j-axis,
35700 REM the locus lies in Quadrant I of the complex plane, forming an
35800 REM approximate parabola which opens to the left. The basic plan is to
35900 REM follow the locus from where it crosses the j-axis at the oscillation
36000 REM frequency, into Quadrant I, and find the point on that locus where
36100 REM the loop gain has a magnitude of 1. The algorithm is as follows:
36200 REM 1. Find the oscillation frequency, 0+jFQ.
36300 REM 2. At this frequency calculate the sign of (AL-1). (AL = magnitude
36400 REM of loop gain.)
36500 REM 3. Increment FQ.
36600 REM 4. For this value of FQ, find the value of SQ for which the loop
36700 REM gain has zero phase.
36800 REM 5. For this value of SQ+jFQ, calculate the sign of (AL-1).
36900 REM 6. If the sign of (AL-1) has not changed, go back to 3.
37000 REM 7. If the sign of (AL-1) has changed, and this value of FQ is
37100 REM within 1Hz of the previous sign-change, exit the routine.
37200 REM 8. Otherwise, divide the FQ-increment by -10.
37300 REM 9. Go back to 3.
37400 REM
37500 REM 1. FIND THE OSCILLATION FREQUENCY, 0+jFQ.
37600 REM
37700 GOSUB 9700
37800 GOSUB 30300
37900 REM
38000 REM 2. AT THIS FREQUENCY, CALCULATE THE SIGN OF (AL-1).
38100 REM
38200 SY% = INT(SGN(AL-1))
38300 IF SY%=-1 THEN STOP
38400 REM ESTABLISH INITIAL INCREMENTATION VALUE FOR FQ
38500 F1 = FQ
38600 DF = (FA-F1)/10#
38700 GOSUB 30300
38800 DE = (FQ-F1)/10#
38900 DF = 0
39000 FQ = F1

```



```

39100 REM
39200 REM 3. INCREMENT FG.
39300 REM
39400 FG = FG + DE
39500 REM
39600 REM 4. FOR THIS VALUE OF FG, FIND THE VALUE OF SQ FOR WHICH THE LOOP
39700 REM GAIN HAS ZERO PHASE. (THE ROUTINE WHICH DOES THAT NEEDS DF = 0,
39800 REM SO THAT IT CAN HOLD FG CONSTANT, AND NEEDS AN INITIAL VALUE FOR
39900 REM DS, WHICH IS ARBITRARILY SET TO DS = 1000.)
40000 REM
40100 DS = 1000#
40200 SQ = 0
40300 GOSUB 30300
40400 IF AL=1! THEN RETURN
40500 REM
40600 REM 5. FOR THIS VALUE OF SQ+JFG, CALCULATE THE SIGN OF (AL-1).
40700 REM 6. IF THE SIGN OF (AL-1) HAS NOT CHANGED, GO BACK TO 3.
40800 REM
40900 IF SYX+INT(SGN(AL-1!))=0 THEN PRINT ELSE 39400
41000 REM
41100 REM 7. IF THE SIGN OF (AL-1) HAS CHANGED, AND THIS VALUE OF FG IS WITHIN
41200 REM 1HZ OF THE PREVIOUS SIGN-CHANGE, EXIT THE ROUTINE.
41300 REM
41400 IF ABS(F1-FG)<1 THEN RETURN
41500 REM
41600 REM 8. DIVIDE THE FG-INCREMENT BY -10.
41700 REM
41800 DE = -DE/10#
41900 F1 = FG
42000 SYX = -SYX
42100 REM
42200 REM 9. GO BACK TO 3.
42300 REM
42400 GOTO 39400
42500 REM
42600 REM
42700 REM *****
42800 REM
42900 REM STEADY-STATE ANALYSIS
43000 REM
43100 REM The circuit model used in this analysis is similar to the one used
43200 REM in the small-signal analysis, but differs from it in two respects.
43300 REM First, it includes clamping and clipping effects described in the
43400 REM text. Second, the voltage source in the Thevenin equivalent of the
43500 REM amplifier is controlled by the input voltage in accordance with an
43600 REM input-output curve defined elsewhere in the program.
43700 REM The analysis applies a sinusoidal input signal of arbitrary
43800 REM amplitude, at the oscillation frequency, to the XTAL1 pin, then
43900 REM calculates the resulting waveform from the voltage source. Using
44000 REM standard Fourier techniques, the fundamental frequency component of
44100 REM this waveform is extracted. This frequency component is then
44200 REM multiplied by the factor  $|Z_L/(Z_L+R_O)|$ , and the result is taken to be
44300 REM the signal appearing at the XTAL2 pin. This signal is then taken to
44400 REM be the signal appearing at the XTAL1 pin. The algorithm is now
44500 REM repeated using this computed XTAL1 signal as the assumed input
44600 REM sinusoid. Every time the algorithm is repeated, new values appear at
44700 REM XTAL1 and XTAL2, but the values change less and less with each
44800 REM repetition. Eventually they stop changing. This is the steady-state.
44900 REM The algorithm is as follows:
45000 REM
45100 REM 1. Compute approximate oscillation frequency.
45200 REM 2. Call a circuit analysis at this frequency.
45300 REM 3. Find the quiescent levels at XTAL1 and XTAL2 (to establish the
45400 REM beginning DC level at XTAL1).
45500 REM 4. Assume an initial amplitude for the XTAL1 signal.
45600 REM 5. Correct the DC level at XTAL1 for clamping effects, if necessary.
45700 REM 6. Using the appropriate input-output curve, extract a DC level and
45800 REM the fundamental frequency component (multiplying the latter by
45900 REM  $|Z_L/(Z_L+R_O)|$ ).
46000 REM 7. Clip off the negative portion of this output signal, if the
46100 REM negative peak falls below zero.
46200 REM 8. If this signal, multiplied by (beta), differs from the input
46300 REM amplitude by less than 1mV, or if the algorithm has been repeated
46400 REM 10 times, exit the routine.
46500 REM 9. Otherwise, multiply the XTAL2 amplitude by (beta) and feed it
46600 REM back to XTAL1, and go back to 5.
46700 REM
46800 REM 1. COMPUTE APPROXIMATE OSCILLATION FREQUENCY.

```

```

46900 GOSUB 9700
47000 REM
47100 REM      2. CALL A CIRCUIT ANALYSIS AT THIS FREQUENCY.
47200 GOSUB 20800
47300 PRINT : PRINT : PRINT "ASSUMED OSCILLATION FREQUENCY:"
47400 GOSUB 26600
47500 PRINT : PRINT
47600 REM
47700 REM      3. FIND QUIESCENT POINT
47800 REM      (At quiescence the voltages at XTAL1 and XTAL2 are equal. This
47900 REM      voltage level is found by trial-and-error, based on the input-
48000 REM      output curve, so that a person can change the input-output curve
48100 REM      as desired without having to re-calculate the quiescent point.)
48200 VI = 0
48300 VB = 1
48400 K1 = 1
48500 VI = VI + VB
48600 GOSUB 13600
48700 IF ABS(VD-VI)<.001 THEN 49200
48800 IF K1+SGN(VD-VI)=0 THEN 48900 ELSE 48500
48900 K1 = SGN(VD-VI)
49000 VB = -VB/10
49100 GOTO 48500
49200 VB = VI
49300 PRINT "QUIESCENT POINT = ";VB
49400 REM
49500 REM      4. ASSUME AN INITIAL AMPLITUDE FOR THE XTAL1 SIGNAL.
49600 EI = .01
49700 NR% = 0
49800 REM
49900 REM      5. CORRECT FOR CLAMPING EFFECTS, IF NECESSARY.
50000 REM      (K1 and K2 are curve-fitting parameters for the ROM parts.)
50100 K1 = (2.5-VB)/(3-VB)
50200 K2 = (VB-1.25)/(3-VB)
50300 IF IC%=2 OR IC%=4 THEN IF EI<(VB+.5) THEN EO = VB ELSE EO = EI - .5
50400 IF IC%=1 OR IC%=3 THEN IF EI<(VB+.5) THEN EO = VB ELSE EO = K1*EI+K2
50500 NR% = NR% + 1
50600 REM
50700 REM      6. DERIVE XTAL2 AMPLITUDE.
50800 VO = 0
50900 VC = 0
51000 VS = 0
51100 FOR NX = -25 TO +24
51200 VI = EO - EI*COS(PI*NX/25)
51300 GOSUB 13600
51400 VO = VO + VD
51500 VC = VC + VO*COS(PI*NX/25)
51600 VS = VS + VO*SIN(PI*NX/25)
51700 NEXT NX
51800 VO = VO/50
51900 V1 = SQR(VC^2+VS^2)/25*FNZM(RL,XL)/FNZM((RL+RD),XL)
52000 REM
52100 REM      7. CLIP XTAL2 SIGNAL.
52200 IF VO-V1<0 THEN VL = 0 ELSE VL = VO-V1
52300 PRINT : PRINT "XTAL1 SWING = ";EO-EI;" TO ";EO+EI
52400 PRINT "XTAL2 SWING = ";VL;" TO ";VO+V1
52500 REM
52600 REM      8. TEST FOR TERMINATION.
52700 IF ABS(EI-V1*B)<.001 OR NR%=10 THEN RETURN
52800 REM
52900 REM      9. FEED BACK TO XTAL1 AND REPEAT.
53000 EI = V1*B
53100 GOTO 50300

```



---

# Design Considerations When Using CHMOS

---

**23**





## CHAPTER 23

# DESIGN CONSIDERATIONS WHEN USING CHMOS

### 23.0 WHAT IS CHMOS?

CHMOS is Intel's n-well CMOS process which is based on the highly developed HMOS-II technology. There are three other types of CMOS processes: p-well, twin-tub, and silicon on sapphire (SOS). All four CMOS structures are discussed in the accompanying article reprint, "Inside CMOS Technology." SOS and twin-tub offer superior performance, but are very costly to manufacture. The n-well technology offers about the same performance as p-well and has been chosen for Intel's microcontrollers because it is more readily adapted to the currently used and well understood HMOS-II technology. This CMOS technology also offers a known path to higher performance products in the future.

Because CMOS tends to have lower gate density and a higher gate count than an NMOS circuit of the same functionality, the ability to scale down the transistor size in CMOS processes is essential to improving the price/performance ratio. The penalty paid for the size reduction, however, is a departure from the traditional CMOS supply voltage range of 3 to 18 volts. CHMOS will be limited to a maximum of 6 volts VCC.

Further reduction in CHMOS die size is accomplished by using dynamic nodes at appropriate points in the circuit, whereas, traditional CMOS is fully static. This reduces the gate count, and therefore, the die size — achieving lower cost. However, the use of dynamic nodes imposes a minimum clock frequency requirement on the CHMOS part.

### 23.1 NOISE IMMUNITY

CMOS noise immunity is greatly overstated. Noise immunity has been described as the amount of noise that can be induced at the input of a gate that will not change the logic state of the output. Noise margin, on the other hand, is the DC levels that will be applied to an input from another output ( $V_{oh}$ ,  $V_{ol}$ ) and the trip point of that input.

On the surface, CMOS would seem ideal for use in noisy environments. Output voltages are rail-to-rail, and input switch points are approximately 50% of VCC. There is one thing wrong with this analysis — CMOS has high impedance inputs. High impedance inputs need only voltage and very little current (nano Amps) to switch its output logic state. TTL, on the other hand, needs voltage and at least 20  $\mu A$  (for an LS device) to switch its output logic state. Because it is a lot more difficult to induce noise in the form of current than in the form of voltage, CMOS tends to be more noise sensitive than TTL.

However, NMOS also has high impedance inputs and has input leakages typically less than 1  $\mu A$ . Because NMOS output voltage swings are considerably less than CMOS,

CMOS has the advantage when inputs and outputs are noisy. Another advantage of CMOS over NMOS is the p-channel pullup instead of a depletion pullup. The p-channel pullup is able to charge up the stray capacitance faster thus signal rise times are significantly improved.

In conclusion, don't fool yourself into thinking that CMOS eliminates the need to be concerned about noise problems. Time is well spent following good design practices and layout techniques from the earliest phases of a project.

### 23.2 LATCH-UP

CHMOS is not immune to traditional CMOS latch-up, but the latch-up threat is highly overrated. Latch-up is usually the result of unforeseen operating conditions, such as an unexpected power-up sequence, inadvertent removal and re-application of a supply voltage, or "hot-socketing" the part (plugging a chip into its socket while the system is active). An energetic voltage spike on VCC or the I/O lines might also trigger latch-up, but a little ringing on the data lines isn't going to cause any problems.

It is helpful to understand the mechanisms involved in the latch-up phenomenon. Figure 23-1A shows the circuit diagram of a typical CMOS output stage. Figure 23-1B shows a plan view of how this stage might look in n-well CMOS (with the gate electrodes of the FETs stripped away, since they are not germane to the discussion). There are two parasitic bipolar transistors in this structure, one pnp and the other npn. The n-well forms the base region of the pnp transistor, which is connected to VCC through the distributed resistance of the n-well. The source and drain of the p-channel pullup FET are dual emitters to the pnp transistor. The p-type substrate is the collector of this transistor and also serves as the base of the npn transistor which is connected to VSS through the distributed resistance of the substrate. The collector of this transistor is the n-well, and the drain and source of the pulldown n-channel FET are dual emitters. These parasitic transistors are shown in Figure 23-1C.

Any pullup FET that shares the same n-well region acts as additional multiple emitters to the parasitic pnp transistor. Much worse, ALL pulldown FETs and input protection devices on the IC act as additional collectors to the parasitic npn transistor. This has several implications. Latch-up is not limited to output stages, inputs only pins and internal gates can also be the cause. So when latch-up does occur, it's hard to tell which device caused it.

In normal operation, both of these parasitic bipolar junction transistors are in an off state, and do not hamper the operation of the FETs. However, if either parasitic junction transistor should turn on, its collector current might turn the other parasitic junction transistor on, and an SCR type effect rapidly ensues which is called latch-up.

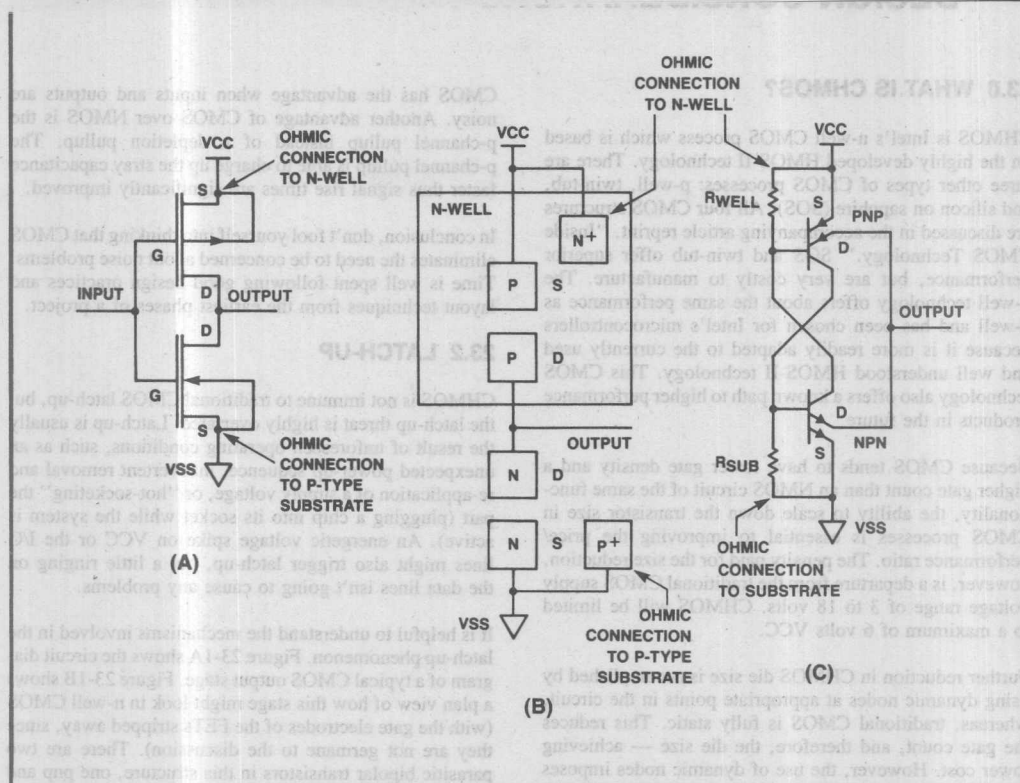


Figure 23-1. CHMOS Inverter

What might turn on one of the parasitic junction transistors? Look at Figure 23-1C. The base of the parasitic pnp is at VCC, and the base of the npn is at VSS. If the output line swings a diode drop above VCC, or below VSS, it forward-biases the "D" emitter of either the pnp or the npn transistor.

Suppose it is the pnp emitter that is forward-biased by the output line swinging a diode drop above VCC. Some of the current that enters that emitter now exits the parasitic device as collector current. It flows down to the juncture of Rsub and the base of the npn transistor. If Rsub is low, the current is shunted through it to VSS, and the npn transistor stays off. If Rsub is high, some of the current crosses the base-emitter junction of the npn, and that transistor will likely turn on.

It is important to note that the designer of the integrated circuit has a certain amount of control over the turn-on probabilities. Grounded guard rings placed around the npn emitters will have the effect of reducing the value of Rsub.

Reducing the value of Rsub has the effect of increasing the amount of current that has to enter the pnp D emitter in order to turn the npn transistor on. The CHMOS, the npn transistor will not turn on if the current entering the pnp D emitter is less than 10 mA.

A similar sequence of events can occur when the output line swings a diode drop below VSS. In this case it is the npn transistor that is in danger of turning on the pnp, depending on the value of Rwell and how much current is involved. Again, the IC designer can reduce the value of Rwell by the judicious placement of guard rings, connected to VCC, around the pnp emitters. When the output line swings a diode drop below VSS, if the current that exits the output pin does not exceed 10 mA, there will be no latch-up.

### 23.3 POWER SUPPLY CONSIDERATIONS

The power supply, as viewed by the microcontroller, should be low in inductance because of the peak currents

associated with CHMOS switching characteristics. Note that the repetition rate of these peak currents increases with the clock frequency. Bypass capacitors of approximately  $0.1\mu\text{F}$  should be used with proper PCB layout to ensure a low inductance, high peak current power source. Low inductance capacitors are also available that fit under the package. These capacitors are also advantages for HMOS in noisy environments (See Application Note AP125 "Designing Microcontroller Systems For Electrically Noisy Environments" in this manual for detailed discussion).

Power supply glitches must be filtered to ensure that the maximum voltage rating of the device is not violated. Violation may induce an SCR effect between VCC/VDD and Vss (latch-up).

The polarity of the power supply must never be reversed. The n-well is connected to VCC and the p-type substrate is connected to ground so normally that pn junction is reversed biased. If VCC or VDD are ever more negative than  $-0.5$  volts with respect to Vss, the n-well/substrate pn junction will be forward biased and short VCC/VDD to ground.

When the microcontroller is powered separately from the surrounding circuitry, the microcontroller should always be powered up before any input signal is applied. The reverse is true when powering down, input signals first then the microcontroller.

When separate VCC and VDD power supplies are used for the 80C49, VDD and VCC must track each other within 1.5 volts (except during power down) and also maintain the 5v 20% specification. This ensures that the CPU, powered by VCC, and the RAM, powered by VDD, have proper voltage levels to communicate. If VCC and VDD cannot be powered up simultaneously, VDD should be applied first.

### 23.4 MINIMIZING POWER CONSUMPTION

The reason CMOS parts draw considerably less current than an NMOS part is that there is no direct path between VCC and ground (refer to figure 10-1A). When the p-channel pullup FET is on the n-channel pulldown FET is off and the output line gets pulled up. The opposite is true, when the n-channel FET is on the p-channel FET is off and the output goes low. There is leakage associated with either the n-channel and p-channel FETs in the off state. The sum of all the leakages from every inverter on the chip is what is called the "quiescent current." This current is in the order of microAmps and is the lesser of the two currents that add up to the total ICC.

When the inverter is switching in either direction, there is a moment in time when both FETs are on creating a low resistance path between VCC and ground. The capacitance on the output line (pin capacitance, trace capacitance, and input capacitance to the next stage) is also charged or discharged which also increases current con-

sumption. This "dynamic current" is a couple of orders of magnitude higher than the quiescent current.

Power supply voltage also comes into play with both types of currents. If VCC is high, the leakage across the off FET is increased. Output capacitance now has to be charged up to a higher level. So a higher VCC also contributes to making ICC larger.

Frequency of switch can also affect the amount of current used. If the inverter stage is switched very slowly, then the average current is equal to the quiescent current. As the frequency of switching is increased, the dynamic current contributes more and more to the total ICC until the dynamic current totally swamps the quiescent current.

Now that we are aware of what components make up the total ICC, we can work on ways of minimizing it. VCC can play a big roll in minimizing power. If the whole purpose in going to CMOS is to minimize heat dissipation and the power is drawn from the 110 volt AC socket, tweak your power supply to the minimum voltage that the system will run. On the other hand if batteries are used you must consider types of batteries available, lifetime of the battery, and the voltage drop off at the end of its life to determine what voltage to start at. Remember the lower the voltage the lower the current draw.

The application also has to be analyzed for what kind of response time is needed to determine how fast the microcontroller needs to run. If the end application is a direct human interface, then the response time can be relatively slow and the processor can run at minimum speed. But if real time decisions need to be made on evaluating incoming data, then the microcontroller needs all the time it can get. The idea is to run the microcontroller as slow as you can and still get the task accomplished.

Intel has added idle mode and power down features to help further manage your power consumption. Idle mode in the 80C51/80C31 stops the clock to the CPU while keeping the oscillator, RAM, interrupts, timer/counter, and serial port alive. Stopping the clock to the CPU decreases the current in the CPU from dynamic to mostly quiescent. Idle mode consumes approximately 1/10th the operating current.

Idle mode allows the microcontroller to minimize its current when no processing needs to be done. The live interrupts, timer/counter, and the serial port can wake the CPU and since the oscillator is running the response time is quick.

Data on the ports are left in the state they were in when idle was invoked. If careful attention is given to the logic state of the port, total system current can be reduced. The state of the port for reduced current is dependent on what the port is driving. If it is a transistor, the transistor should be put into its off state. If the port pin is driving another CMOS gate, the loading of the port pin would be minimal and a choice of the logic state may be made on what is



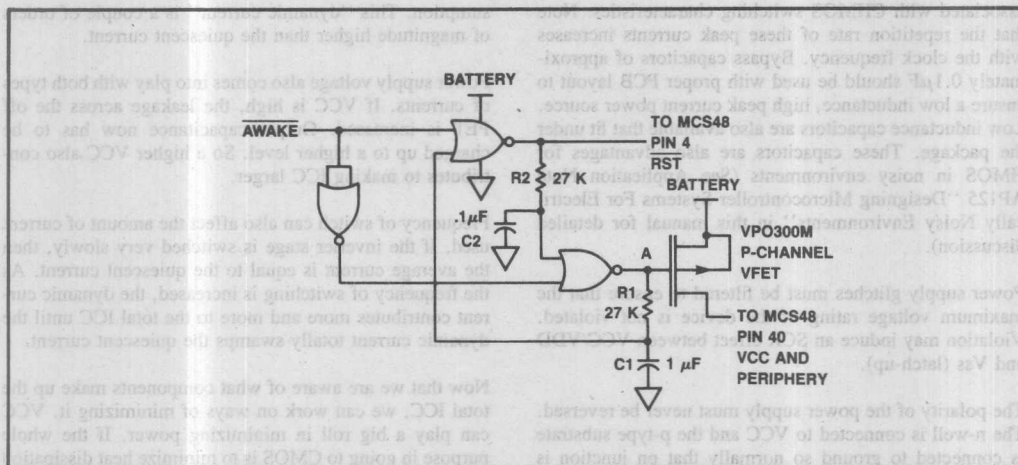


Figure 23-2. MCS®-48 Power Down Circuit

on the outputs of the CMOS gate. If TTL is being driven, the reduced current would be when the port pin is high. A little thought in this area can go a long way in minimizing system current.

Power down removes all internal clocks to decrease the current to totally quiescent current while the contents of the RAM are saved. This mode is useful in hand held applications where data needs to be saved between uses. On the 80C51, the port pins are left in the same state they were in when the power down mode was called. The same thought processes that were needed for reduced current in the idle mode are valid for power down also.

Figure 23-2 shows a simple circuit that uses one quad NOR CMOS integrated circuit and some external resistors and capacitors to power down the CHMOS MCS48 family under the control of one input. To activate the operation of the microcontroller, the AWAKE/ signal is pulled low and in turn, node A goes low turning on the p-channel VFET. This allows VCC to the microcontroller to be pulled to the power supply minus V<sub>sd</sub>. Node A being low brings RESET high after a time determined by the R1C1 time constant. The RC has been chosen to allow 10 mS between VCC going high and RESET going high to allow the oscillator time to stabilize.

Powering down is accomplished by AWAKE/ going high which pulls RESET low. The time delay R2C2 allows the reset signal enough time to signal the microcontroller to save the RAM before VCC is shut off.

When idle mode and power down are used in conjunction with slow operating speed it can reduce your power needs to a minimum.

Unused input only pins on the MCS48 family such as SS/, T0, and T1 should not be allowed to float. The inputs would float about the trip point of the input buffer. This switching of the input buffer can waste up to .5 mA per input. Tie all unused input only pins high or low.

When the CHMOS units are being used with external program or data memory, PortO (Data Bus on the MCS48 products) is left floating when in idle mode. PortO also floats on the 80C51 when in the power down mode. The same condition as described above can happen with the input gate on those pins. Without tying these pins high or low at least .5 mA × 8 or 4 mA could be wasted. Tie the PortO pins high or low through a 500KΩ or 1MΩ resistor. A little extra board space is minimal when considering the extra power savings.

The quasi-bidirectional pins have internal pullups so the inputs on these pins never float.

## 23.5 CHMOS I/O PORT STRUCTURE

The CHMOS I/O ports have similar drive capability to their HMOS counterparts, but the differences must be noted.

### 23.5.1 As An Output Pin

The I/O port structure is implemented as shown in Figure 23-3.

As an output pin latched to a low (0) state, pullups P1, P2, and P3 are in an off state while the pulldown N1 is on. This configuration uses little current, since there is no path between VCC and ground in the output buffer stage.

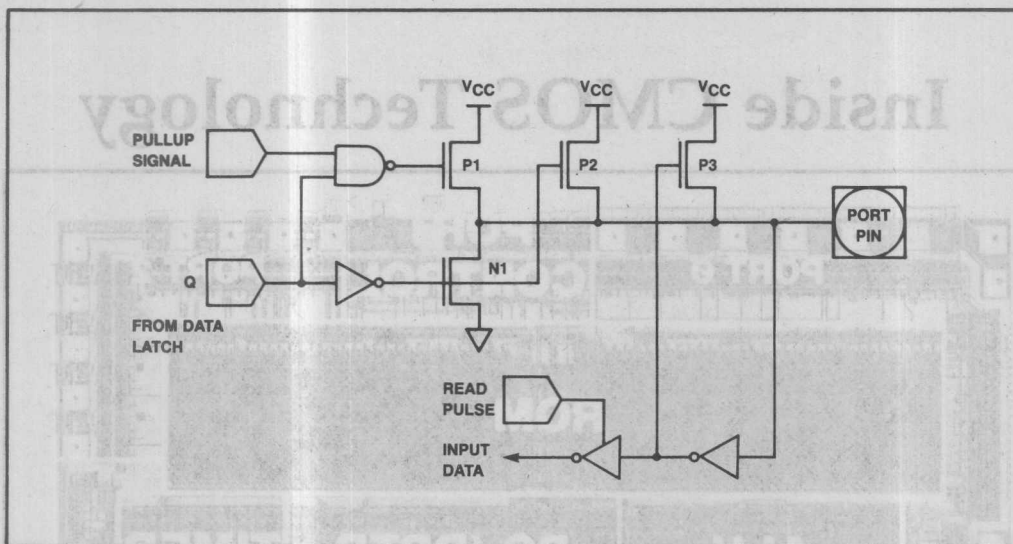


Figure 23-3. CHMOS Quasi-Bidirectional I/O Port Structure

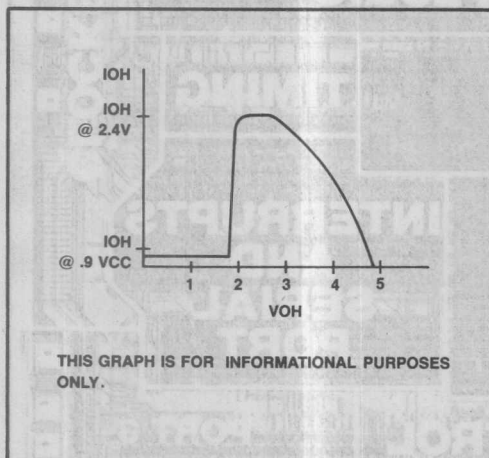


Figure 23-4. CHMOS I/O Current Characteristics

When the output pin is to be latched in the high (1) state, the data line turns N1 off and turns on the weak ( $5\mu\text{A}$ ) pullup P2. At the same time P1 (a strong pullup) turns on for one state time pulling the output up very quickly. Once the output voltage is above approximately 2 volts, P3 turns on to supply the source current. P3 and the inverter form

a latch. This latch along with the support of P2 keeps the output high. Figure 23-4 shows the  $\text{VOH}$  vs  $\text{IOH}$  curve for this output structure when P1 is off.

## 23.5.2 As An Input Pin

To use the I/O port as an input, a one must be written to the pin first, leaving the pin in a high state. When the input goes below 2 volts, P3 turns off to avoid any high sink currents from being presented to the input device. Note when returning back to a one, P2 is the only internal pullup that is on. This will result in a long rise time if P2 is the only pullup.

## 23.5.3 Interfacing Between CHMOS and Other Logic Families

Interfacing Intel's CHMOS to other logic families is very simple and straight forward. When VCC is kept within 10% of 5 volts all inputs (except those noted in the data sheets) and outputs are TTL compatible. CMOS compatibility is achieved as long as VCC is kept within 20% of 5 volts.

When driving a high current load, the output current ( $\text{IOH}$ ) must be limited to keep the output voltage ( $\text{VOH}$ ) at a minimum of 2 volts. If the voltage is pulled below 2 volts the output current will be dropped to approximately  $5\mu\text{A}$ . See Figure 23-4.

# Inside CMOS Technology

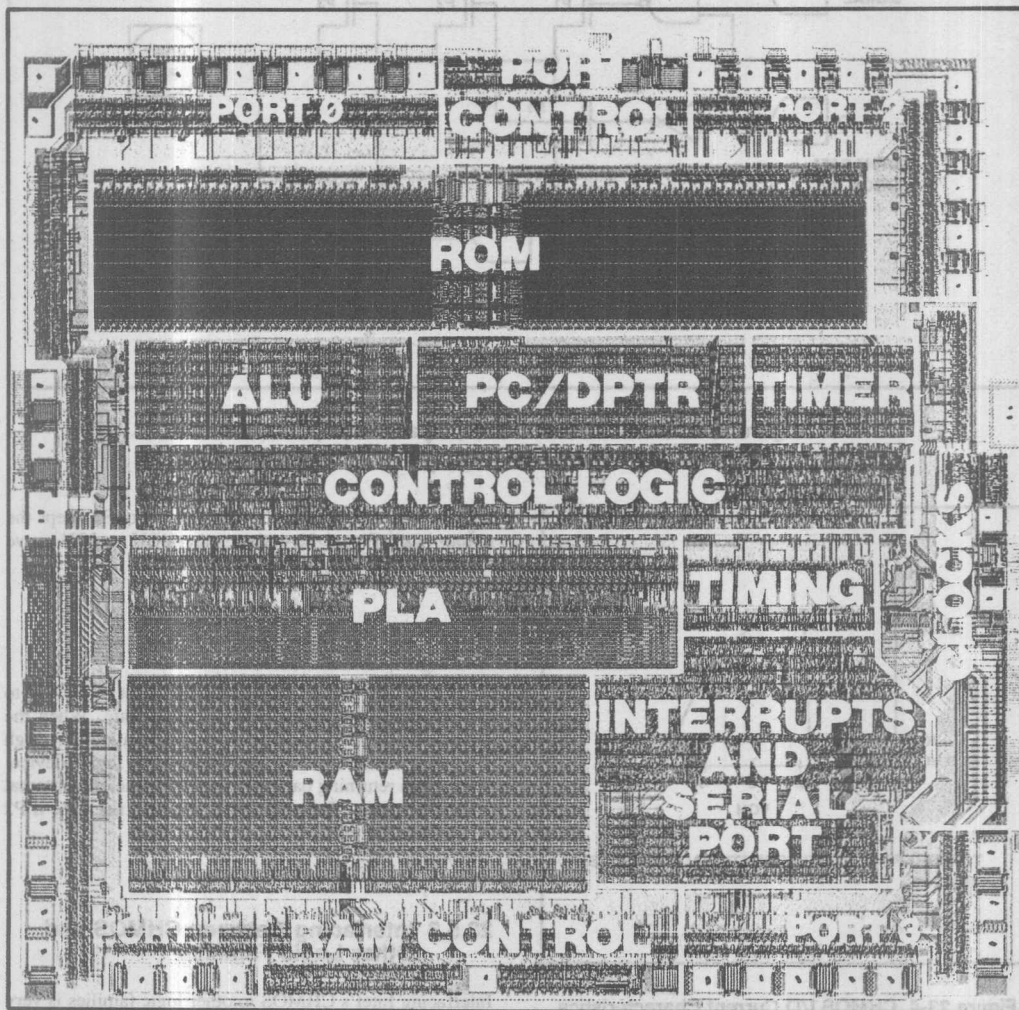


Photo 1: The die for the 80C51, with the functions of the various sections identified.

*How CMOS devices are manufactured and a look at three of them*

by Martin B. Pawloski, Tony Moroyan, and Joe Altnether

Order Number: 230860-001



CMOS (complementary metal-oxide semiconductor) has often been called the ideal technology. It has low power dissipation, high immunity to power-supply noise, symmetric switching characteristics, and a large supply-voltage tolerance. But CMOS has rarely been used for advanced VLSI (very-large-scale-integration) microcomputer designs. Because of the complexity of the CMOS process, the ICs (integrated circuits) produced have traditionally had a relatively poor price/performance ratio.

As a result, CMOS was used only in applications that required low power and were neither performance conscious (such as in calculators and watches) nor cost conscious (many military applications, for example). Suddenly, however, all major semiconductor companies have announced either advanced CMOS products or the intention of designing their next generation of high-performance microprocessors using CMOS technology.

What has happened to make CMOS both affordable and high performance? For one thing, the dominant VLSI technology, NMOS (n-channel metal-oxide semiconductor), is rapidly approaching the process complexity of standard CMOS. It is not unusual nowadays for NMOS technology to have up to four transistor types with different operating characteristics. Much of the complexity of this process is added simply to help VLSI designers keep the operating power of their circuits under control.

Second, CMOS circuit designers are being more selective in the use of static CMOS logic. Critically placed dynamic logic, creative circuit design, and use of modes that offer varying degrees of power consumption are all tricks designers are using to maintain the advantages of CMOS.

Finally, aggressive reduction in CMOS transistor size is being used to bring CMOS performance in line with that of NMOS. As a matter of fact, many manufacturers are developing CMOS as a derivative of their advanced NMOS processes.

This not only improves CMOS performance levels but also boosts reliability and reduces development costs.

### **The Evolution of LSI**

Early LSI circuits were built with p-channel MOS transistors, which permitted high-circuit densities yet were relatively slow and difficult to interface to normal integrated circuits, such as TTL (transistor-transistor logic). As an example, the 1103-type 1K by 1-bit dynamic RAM (random-access read/write memory), circa 1971, required its inputs (address, controls, and data) to swing between 1 and 15 volts (V) although its output was measured in millivolts — hardly TTL compatible! About 1974, NMOS came to the rescue. It provided faster speed, and most of its inputs and outputs were TTL compatible.

---

### **Low power requirements are a major advantage of designing a system that uses CMOS.**

---

NMOS was more difficult to manufacture than PMOS because contaminants would vary the thresholds of the n-channel transistors, causing deviations in speed and performance. But this problem was quickly overcome through ultraclean processing rooms, and NMOS became the workhorse technology because it cost less to manufacture, was easy to use, and had good speed-power characteristics. And NMOS technology had potential for greater improvement of its speed-power characteristics through scaling (or shrinking) of the silicon devices. The result of this scaling was HMOS (high-speed NMOS), which accomplished three objectives: increased speed, reduced power, and increased density.

Over the past 10 years, the reduction in transistor size has, at the device level, increased memory density by a factor of 64, increased

speed by a factor of 3, and reduced power consumption by a factor of 100. However, the scaling cannot continue ad infinitum because of resolution limitations of the photolithographic equipment used to make the circuits as well as breakdown mechanisms within the devices. More important, even before these limitations are reached, heat dissipation will prohibit major enhancements with NMOS. Heat generation increases exponentially with transistor count, and, at densities approaching 150,000 transistors per integrated circuit, special cooling measures are required. This heat can accelerate failure mechanisms within the silicon, reducing device and system reliability. To hurdle this barrier, low-power devices must be used.

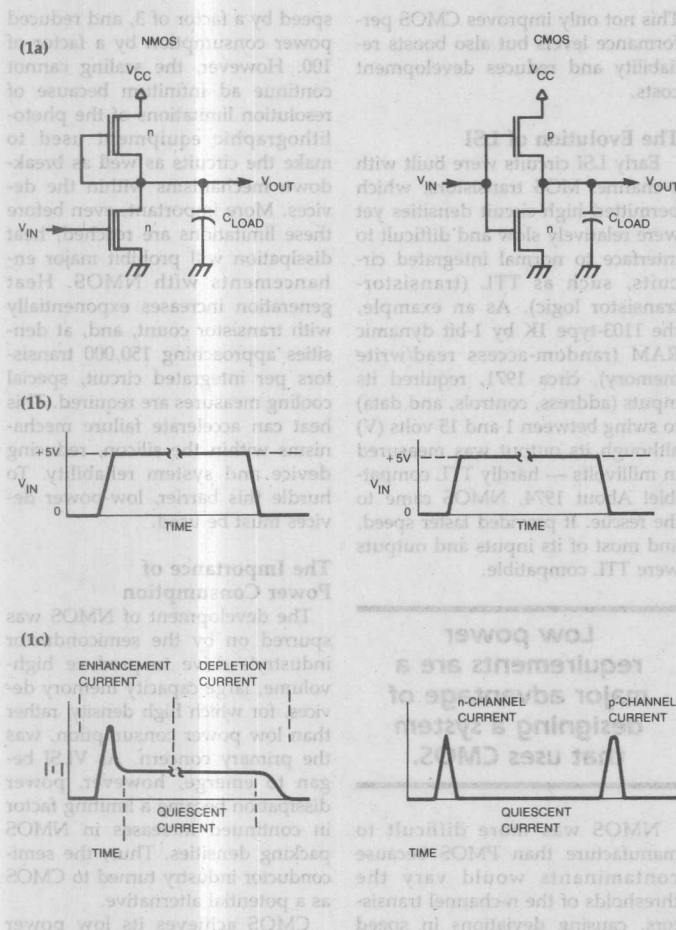
### **The Importance of Power Consumption**

The development of NMOS was spurred on by the semiconductor industry's drive to produce high-volume, large-capacity memory devices, for which high density, rather than low power consumption, was the primary concern. As VLSI began to emerge, however, power dissipation became a limiting factor in continued increases in NMOS packing densities. Thus, the semiconductor industry turned to CMOS as a potential alternative.

CMOS achieves its low power dissipation through the use of both p- and n-channel transistors (hence the name "complementary"). Essentially, no DC power is dissipated in either logical state, and AC power occurs only during the relatively short switching period. Because most circuitry in a complex design is active only 10 to 20 percent of the time, CMOS achieves a dramatic reduction in power dissipation compared with NMOS, which continually dissipates DC power whenever an operating voltage is applied.

Low power requirements are a major advantage of designing a system that uses CMOS. Reducing power requirements has a domino effect that often substantially reduces the cost of the end product:





**Figure 1:** A comparison of NMOS and CMOS technologies. Figure 1a shows the schematic diagrams of an inverter as implemented in both NMOS and CMOS. A hypothetical input waveform and the resulting transistor currents are shown in 1b and 1c.

- low power allows smaller, lower-cost power supplies to be used
- power distribution in the system is simplified
- cooling fans can be eliminated
- printed-circuit boards can be packed more densely and can thus become smaller

With smaller power supplies, denser circuit boards, and no fans, smaller cabinets can be used, resulting in savings in chassis and enclosure costs. Also, power fail-safe

and hand-held use become possible if battery operation is feasible.

### Basic CMOS Operation

To truly understand the promises (and problems) facing both the CMOS VLSI digital designer and the CMOS systems designer, one must first understand some CMOS fundamentals.

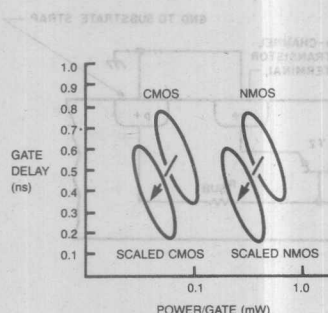
Figure 1 compares the circuit diagrams and current characteristics of both an NMOS and a CMOS inverter. The NMOS inverter uses an n-channel depletion-mode transis-

tor as the pull-up device (which drives the output line high) and an n-channel enhancement-mode transistor as the pull-down device (which drives the output line low). The pull-up transistor is used as a load; its operation approximates that of a constant current source. The pull-down transistor is used as the switching device; when active, it discharges the load, and when inactive it lets the pull-up charge the load. MOS loads are primarily capacitive and include the parasitic capacitances of the inverter itself, interconnect capacitances, and the thin-oxide capacitances of all the gates the inverter is driving.

Let's note several characteristics of an NMOS inverter. When the pull-down device is turned on, it not only has to sink the current from the capacitive load, but it also has to sink the current supplied by the pull-up load device. Even in the quiescent state this current component from the pull-up device still exists. Because logic gates spend most of their time in the quiescent state, this quiescent current accounts for up to 90 percent of the total power dissipated in NMOS VLSI designs; the remaining 10 percent is switching or dynamic power.

A second related characteristic is that the inverter's output voltage in the low state,  $V_{OL}$ , is dependent on the ratio of the impedances of the pull-down and pull-up devices. This ratio affects the noise margin and switching speed and is generally around 4:1. Such a ratio results in a  $V_{OL}$  on the order of 0.2 V to 0.3 V. It also causes asymmetric switching characteristics: the fall time of the inverter is significantly faster than its rise time.

The CMOS inverter uses a p-channel enhancement-mode transistor as the pull-up device and an n-channel enhancement-mode transistor as the pull-down device. In a CMOS inverter, both the pull-up and pull-down transistors are used as switching devices. When the input changes from low to high, the p-channel device shuts off and the n-channel transistor discharges the load. When the input changes from



**Figure 2:** The speed versus power consumption characteristics of NMOS and CMOS technologies. Note the advantages gained by scaling (reducing the size) of the integrated components.

high to low, the n-channel device shuts off and the p-channel transistor charges the load. While almost all current from the CMOS inverter is used to charge or discharge the load, a small current component does not flow through the load. This is a result of the fact that both the p-channel and n-channel transistors are on for a short period of time during the input voltage transition. This current component is typically less than 10 percent of the total inverter current, though it depends greatly on the rise and fall times of the input signal.

With no quiescent power component, a CMOS inverter's dynamic power dissipation represents only a small fraction of an equivalent

NMOS inverter's power dissipation. Also, the CMOS inverter is a "ratio-less" design, having only one transistor active after an input transition. This lets  $V_{OL}$  go all the way to ground potential, resulting in better noise tolerance than NMOS inverters. It is also a simple matter to design CMOS circuits with outputs that have equal rise and fall times. While this is important in some circuits, it is generally not taken advantage of in VLSI designs because it requires greater chip area.

For NMOS and CMOS technologies with similar transistor dimensions and gate oxides, gate delays are essentially identical. The speed-power products for such a set of NMOS and CMOS technologies are shown in figure 2. This graphically illustrates the tremendous power advantage CMOS offers when used in high-performance VLSI designs.

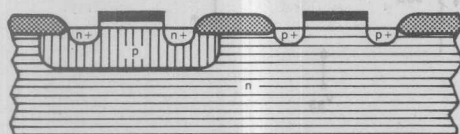
While CMOS enjoys significant electrical advantages over NMOS, it does have a cost disadvantage. One small factor is the larger number of process steps needed to fabricate a CMOS device. More significant is the larger die required because CMOS has lower gate density.

### CMOS Technologies

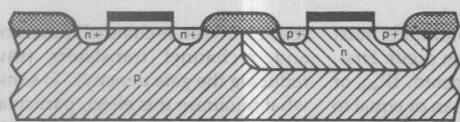
Figure 3 shows the four major CMOS technologies in use today: p-well bulk, n-well bulk, twin-tub bulk, and silicon-on-sapphire (SOS). P-well CMOS uses a p-type diffusion into an n-type bulk silicon substrate to form an n-channel transistor. The p-channel transistor is built directly in the bulk. This is the original CMOS technology, which has many years of good performance and reliability behind it.

The n-well CMOS process starts with a p-type substrate. N-type material is diffused into it to form the n-well in which p-channel devices are built. N-channel devices are built directly in the bulk substrate. An n-well CMOS process is usually derived from an advanced NMOS process. It also permits a highly optimized n-channel transistor, which yields a slight performance advantage over a p-well CMOS process.

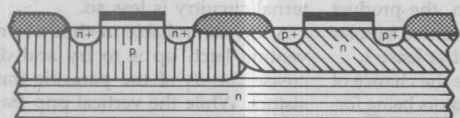
(3a) p-well (original)



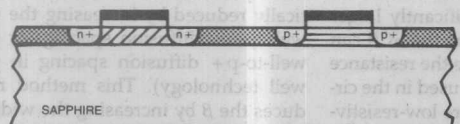
(3b) n-well



(3c) twin tub



(3d) SOS (no latchup)



**Figure 3:** Cross sections of transistors formed by each of the four major CMOS processes. Figure 3a is a p-well bulk CMOS transistor; figure 3b shows an n-well bulk device; figure 3c is an example of a twin-tub bulk CMOS transistor; the transistor in figure 3d is formed using silicon-on-sapphire technology.

Twin-tub CMOS combines n-well and p-well technologies by diffusing both an n-well for the p-channel transistor and a p-well for the n-channel transistor. The twin wells are usually formed in a lightly doped n-type substrate. While it is a slightly more complex and costly process than either n-well CMOS or p-well CMOS, twin-tub CMOS has the advantage of being able to optimize the performance of both the n-channel and p-channel devices. Thus, this process gives the highest overall performance of the bulk CMOS technologies.

The highest performing CMOS technology is SOS. Silicon islands are grown on an insulating sapphire substrate. N-channel or p-channel transistors are then built on the islands. High performance is achieved due to the significant reduction of parasitic capacitance. SOS also offers good gate density because no parasitic bipolar transistors are around to cause a phenomenon called *latch up*. Unfortunately, SOS devices are difficult and expensive to manufacture. For example, unused sapphire wafers cost approximately 10 times more than bulk silicon wafers.

While CMOS suffers a cost penalty of about 20 percent due to process differences, it generally suffers more significantly because of die size. (While processing steps have a linear relationship with cost, die size has an exponential relationship.) CMOS dies are larger than equivalent NMOS designs even when aggressive transistor scaling is employed. Three major factors contribute to this: the area used in trying to prevent latch up, CMOS logic-gate structure, and static design techniques.

#### Latch Up Prevention

Bulk CMOS technologies have parasitic bipolar transistors that, if improperly biased, can cause a phenomenon called latch up. This potentially destructive action results from triggering an SCR (silicon-controlled rectifier) formed by the transistors and can cause extremely large currents to flow. Figure 4 shows the construction of the parasitic SCR in an n-well bulk CMOS device.

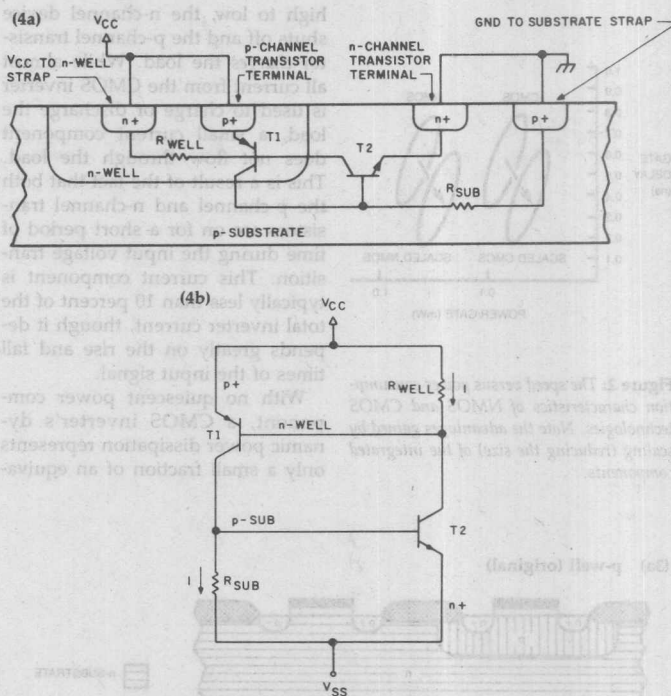


Figure 4: Parasitic SCR in bulk CMOS can cause latch up. Figure 4a shows how the parasitic transistors are formed in the silicon; figure 4b is a diagram of the equivalent circuit.

Two well-defined conditions must exist before latch up can occur. First, for the SCR to be triggered,  $IR_{well}$  or  $IR_{sub}$  must be greater than or equal to 0.7 V. This forward-biases the base-emitter junctions of the parasitic bipolar transistors. Second, to sustain the latch up condition, the product of the  $\beta$ s (gains) of the two bipolar transistors must equal at least 1.

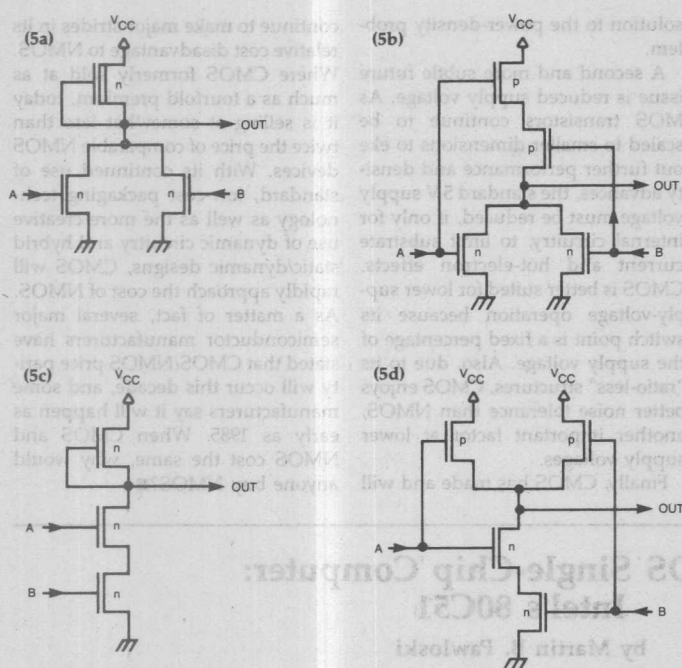
In order to minimize the chance of one of the SCR's transistors being forward-biased, every attempt is made to reduce the resistance values as much as is feasible. This has the effect of requiring significantly larger injected currents before the SCR can be triggered. To reduce the resistance values, *guard rings* are used in the circuitry. (Guard rings are low-resistivity connections to the supply voltages placed around the CMOS p-channel and n-channel transistors.) While guard rings reduce the SCR bias resistor values, they also increase the

space between n-channel transistors and p-channel transistors (thus reducing the gate density). To somewhat minimize this effect, particularly sensitive areas (like VLSI component's I/O pins) are heavily guard ringed, while the more protected internal circuitry is less so.

A less controllable method of preventing latch up is to try and decrease the  $\beta$ s of the parasitic transistors. While the vertical pnp transistor's  $\beta$  is set by the process design, the lateral npn transistor is more directly controllable. Its  $\beta$  can be drastically reduced by increasing the n-well-to-n+ diffusion spacing (or p-well-to-p+ diffusion spacing in p-well technology). This method reduces the  $\beta$  by increasing the width of the transistor's base. While this is an effective way of decreasing the gain of the parasitic structure, it also reduces the gate density.

In bulk CMOS technologies, to





**Figure 5:** A comparison of typical logic gates in NMOS and CMOS form. Figure 5a is an NMOS NOR gate, while figure 5b is a CMOS version; figure 5c is an NMOS NAND gate, and figure 5d is a CMOS version.

give absolute protection against latch up is not only tremendously expensive in silicon area, but it is also virtually impossible. CMOS designers sacrifice area to ensure there is enough margin in their design to protect it from latch up in normal operating-system environments.

### Logic-Gate Structures

Gate densities are also reduced in CMOS because standard CMOS logic gates are built from more transistors than their NMOS equivalents. Standard CMOS logic-gate design has a 1:1 ratio of n-channel transistors to p-channel transistors. For example, the two input gates shown in figure 5 take four transistors in CMOS and only three in NMOS. The relative density decreases as the number of inputs increases. For example, three-input gates require six transistors in CMOS and only four in NMOS; four-input gates require eight transistors in

CMOS and only five in NMOS, etc. As a matter of fact, it is rare to have a standard CMOS gate with more than three inputs because the self-loading and the transistor stack make the structure inefficient in both speed and area. On the other hand, it is not unusual in NMOS to have gates with as many as eight inputs.

### Static Design Techniques

A final reason for the lower CMOS gate densities is the use of static logic (modern VLSI NMOS microcomputer designs rely heavily on dynamic circuitry). Dynamic circuitry essentially uses a small capacitor as a latch to store logic values. This technique saves both area (by reducing the number of transistors in a gate) and power (by reducing the number of gates in structures like latches, flip-flops, shift registers, etc.). Employing dynamic design can reduce an NMOS latch's

area by 30 percent and its power consumption by 50 percent. However, the problem with dynamic circuitry is that the capacitor used to store the logic value is leaky and will, over time, discharge and lose its data. This is the same problem faced by dynamic memory designers. The solution is to periodically refresh the capacitor, which forces a minimum operating frequency to be adhered to.

CMOS can also use dynamic circuitry, especially to increase the ratio of n-channel transistors to p-channel transistors. Because static CMOS designs have a 1:1 ratio of n-channel to p-channel transistors, being able to increase this ratio will have the effect of giving CMOS a higher gate density (but the minimum operating-frequency characteristic of dynamic circuitry often conflicts with the CMOS potential of absolutely minimizing power). Therefore, while true static CMOS design does give the lowest possible power consumption (by allowing the device to operate at frequencies all the way to DC), dynamic CMOS designs, being more dense and resulting in smaller die sizes, tend to be more cost-effective. Thus, two trends are developing in the use of CMOS for VLSI microcomputer design.

Designers of the next generation of 16- and 32-bit microprocessors are choosing CMOS. Here, the goal is not to operate at the lowest possible power level but rather to keep the operating power under a maximum level for cooler junction temperatures, higher performance levels, and the ability to use standard low-cost packages. In these designs, extensive use is made of dynamic logic. The ratio of n-channel transistors to p-channel transistors is often as high as 3:1.

Designers of 4- and 8-bit single-chip microcomputers are choosing CMOS to accommodate a host of new portable, hand-held, and ultra-low-power applications. Here, the goal is to minimize the operating power levels consistent with the performance required by the application. In the simpler micro-



computers, true CMOS static logic is used—their simpler structure still allows a relatively small die size, while the low-performance applications they are appropriate for allow low operating frequencies. On the other hand, the more complex, higher-performance, single-chip VLSI components still make maximum use of static logic but are forced into dynamic logic for large arrays to keep the die cost down.

#### Future CMOS

CMOS will be the technology of choice for VLSI microcomputer designs. For one thing, with the advent of hundreds of thousands of transistors on a die, CMOS is the only technology that offers a cost-effective

solution to the power-density problem.

A second and more subtle future issue is reduced supply voltage. As MOS transistors continue to be scaled to smaller dimensions to eke out further performance and density advances, the standard 5-V supply voltage must be reduced, if only for internal circuitry, to limit substrate current and hot-electron effects. CMOS is better suited for lower supply-voltage operation because its switch point is a fixed percentage of the supply voltage. Also, due to its "ratio-less" structures, CMOS enjoys better noise tolerance than NMOS, another important factor at lower supply voltages.

Finally, CMOS has made and will

continue to make major strides in its relative cost disadvantage to NMOS. Where CMOS formerly sold at as much as a fourfold premium, today it is selling at somewhat less than twice the price of comparable NMOS devices. With its continued use of standard, low-cost packaging technology as well as the more creative use of dynamic circuitry and hybrid static/dynamic designs, CMOS will rapidly approach the cost of NMOS. As a matter of fact, several major semiconductor manufacturers have stated that CMOS/NMOS price parity will occur this decade, and some manufacturers say it will happen as early as 1985. When CMOS and NMOS cost the same, why would anyone buy NMOS?■

## A CMOS Single-Chip Computer: Intel's 80C51

by Martin B. Pawloski

Intel's 80C51 is an interesting example of how the static logic versus dynamic logic trade-off was made in an actual product design. The 8051 is an 8-bit, single-chip microcomputer with 4K bytes of ROM, 128 bytes of RAM, two 16-bit counter/timers, multilevel interrupt control, 32 I/O pins, full-duplex UART (universal asynchronous receiver/transmitter), and on-chip oscillator and clock circuits. A die with the sections identified by functions is shown in photo 1 on page 94.

The CMOS version of the 8051, called the 80C51, is targeted at a number of applications that require both high performance and low power consumption. In areas like telephony, automotive control, industrial control, and portable instrumentation, the 80C51 operates at or near its maximum speed, even if only for short intervals. (For example, most real-time applications need an external-interrupt response time of less than 100 microseconds ( $\mu$ s); more demanding applications require better than 10- $\mu$ s response. While the response must be quick, and the interrupt routine executed quickly; the

processor spends a significant portion of its time idle.)

Once the performance requirements of the application are known, it is possible to specify a minimum operating frequency. For the 80C51, a hybrid static/dynamic design was proposed that allows a minimum die size and includes various modes of operation to minimize power consumption.

First, the only areas of the design that were made dynamic were the (very large) ROM and Control arrays. These arrays contain almost 50,000 transistors and constitute a major portion of the die. By making them dynamic, an area savings on the

order of 40 to 50 percent was accomplished.

Second, the processor, all the peripheral functions, the RAM, and the I/O ports were made static. This allowed two modes of operation other than normal operating mode: *Idle* mode and *Power Down* mode.

Because in many applications the processor does nothing more than wait for an event to happen, in the *Idle* mode the major clocks of the device are stopped and only smaller ancillary clocks operate to drive the peripheral counter/timers, external-interrupt control, and the serial channel. When one of the peripherals generates an interrupt, the processor

Product	Technology	V <sub>cc</sub> Supply Voltage Range	Operating-Frequency Range	Normal Operating Mode (I <sub>cc</sub> Max)	Idle Mode (I <sub>cc</sub> Max)	Power Down Mode (I <sub>cc</sub> Max)
80C51	CMOS	4-6 V	12 MHz Max 1.2 MHz Min	24 mA 2.4 mA	3 mA 0.3 mA	50 $\mu$ A 50 $\mu$ A
8051	NMOS	4.5-5.5 V	12 MHz Max 1.2 MHz Min	150 mA 130 mA	— —	20 mA 20 mA

Table 1: A comparison of the CMOS and NMOS versions of the 8051.

clocks are restarted and instruction execution resumes in the interrupt-service routine. The Idle mode reduces power consumption by almost an order of magnitude.

In Power Down mode, all the clocks inside the device are shut off and only the internal 128 bytes of RAM are "kept alive." The only current consumed is a minute amount due to pn-junction leakage. Static logic was designed in the peripheral sections in order to support this mode because

no clocks are available to refresh dynamic logic. In both the Idle and Power Down modes, special provisions are made for the dynamic circuits in the ROM and Control areas to enter a pseudostatic condition that prevents any extraneous power consumption due to voltage drift on capacitive storage nodes.

Table 1 compares the NMOS 8051 to the CMOS 80C51. The 80C51, designed in Intel's HMOS-derived n-well process called CHMOS, is less

than 10 percent larger than the NMOS design and consumes only 15 percent of the normal operating power. More significant power savings are possible by operating the 80C51 at lower frequencies or by using the Idle mode.

*Martin B. Pauloski (5000 West Williams Field Rd., Chandler, AZ 85224) is involved in the product planning, definition, and implementation of both NMOS and CMOS single-chip microcomputers at Intel Corp.*

## A Look at CMOS Dynamic Memory

by Joe Altnether

The fast-growing portable-computer market is placing severe demands on semiconductor memory. For optimum system performance, these components must limit their power dissipation to suit battery operation and backup, and they must achieve the high data bandwidths and increased speeds needed for fast processing and high-resolution graphics. As the market reaches a projected \$4.8 billion level by 1987 (a tenfold increase over 1982 levels), these requirements will combine to fuel the use of high-performance CMOS dynamic RAMs.

One architecture that can increase the speed of a CMOS dynamic RAM incorporates static-column address decoders: static circuits perform the selection of the column address of the RAM. Previously, this architecture has not been used with dynamic RAMs because of the increased power consumption of the static circuits over that of the dynamic circuits, and the advantage of low power consumption would have been lost. But with CMOS, the increased power consumption is negligible.

### Memory-device Architecture

RAMs are organized internally as rows and columns of storage cells. Data access occurs at the intersection of a row address and a column address. In dynamic RAMs, the row

and column addresses are multiplexed to reduce package size and pin count: the row addresses are clocked into the device with the RAS (row address strobe) signal, causing one row of data (1 bit from each of the 256 columns in a 64K-bit dynamic RAM) to be fed into the 256 internal sense amplifiers. (Because of the low internal signal levels, each column must have an associated sense amplifier to sense and restore memory-cell data.) Next, column addresses are presented to the device and clocked into it with the CAS (column address strobe) signal. These column addresses are then decoded to select one of the 256 bits. Faster access and cycle times are obtained within a row (or "page") after the first access to it because the 256 bits within the row continue to reside in the sense amplifiers and need not be refetched. Reapplying only column addresses, then, in what is known as *Page Mode* operation, provides fast serial accesses and can increase cycle times by a factor of 2.

The CMOS dynamic RAM can incorporate static-column circuits to provide performance equivalent to that of high-speed static RAMs. With CMOS, the static-decoding circuits reduce the internal number of clocks by a factor of 3, eliminating the need to allow for setup and hold times of signals with respect to clocks and the need to compensate for timing skews

due to process variances. With static-column circuits, precharge times are drastically reduced (in Page Mode operation of the Static-Column-mode device, precharge time is reduced from 30 nanoseconds [ns] to 5 ns). This precharge time reduction and the faster access times typically increase the memory's bandwidth to 20 MHz. (Performance of memory discussed here is based on the experimental 64K-bit CMOS dynamic RAM that Intel presented at the ISSCC conference in February 1983.)

With static-column architecture, two different types of Page Mode operation are possible: Static Column mode and Ripplemode. Static Column mode uses the RAS line and row addresses in the conventional manner, but once the row has been selected, data can be accessed merely by changing column addresses. As with a static RAM, column addresses must remain stable and valid for the entire address access cycle. Access time is measured from column addresses rather than the occurrence of CAS. (Typically, access from column addresses is 30 ns; from CAS, it is 10 ns.)

In operation, CAS is used to place the output in a high-impedance state or to activate an output buffer. CAS can be held active during the entire page cycle. In fact, it is possible to keep CAS permanently active (i.e., grounded). During a write cycle,

however, addresses as well as data are latched by CAS or WE, whichever occurs last. Operation is identical to that of an NMOS dynamic RAM in this case. This action ensures that the data is written into the proper memory location.

Although Static Column mode provides fast, easy accesses, speed at the system level is limited by how fast addresses for the next cycle become valid; the time to generate and stabilize the addresses must be added to the cycle time. Increased system speed can be obtained by using Ripplemode. With this mode, static-column circuits are again used to obtain access from valid column addresses, but the addresses are latched on the falling edge of CAS, removing the requirement for addresses to remain valid throughout the entire cycle. As a result, during the current cycle addresses for the next cycle can be set up or pipelined.

Column addresses enter the RAM through the internal address latch. This latch, controlled by CAS, provides flow-through operation. When CAS is inactive, the latch is open, and addresses pass through continuously to the static-column decoders. Any change in address is transmitted immediately to the decoder. Consequently, access to the RAM is again measured from valid column addresses. The latch captures the current address on the fall of CAS, permitting the system address to change while the access occurs. CAS also serves as an output enable on the data output. Static Column mode and Ripplemode both permit continuous data streams up to 20 MHz.

CMOS technology and static-column architecture provide more than low power consumption and high bandwidth. In addition, static-column decoding simplifies system design by eliminating critical timing relationships while providing higher system speed. Access from column addresses gives usable speed for single random accesses within the RAM. Also, the CMOS technology enhances reliability by incorporating a mechanism to significantly reduce soft errors. Finally, increased stored charge creates larger internal signal

levels, which can more easily be differentiated from noise. As a result, the CMOS dynamic RAM has wider operating margins and system reliability is improved.

### Power Consumption

At the system level, dynamic memory has three components of power: active, standby, and refresh. The system's power consumption is defined as

$$P = V(MI_A + KI_S + NI_R)$$

where  $P$  = system power,  $V$  = voltage (5.5 V worst case),  $I_A$  = active current,  $I_S$  = standby current,  $I_R$  = refresh current,  $M$  = number of active devices,  $K$  = number of devices in standby, and  $N$  = total number of devices.

CMOS reduces the first term, the active current, relative to NMOS by a factor of 2. In addition, the lower active current reduces supply voltage transients, thus simplifying printed-circuit-board design and reducing decoupling-capacitor requirements.

The second term, standby current, is also reduced by a factor of 2 at TTL input levels. Driving the RAS signal to a CMOS level ( $V_{DD} - 0.5$  V) places the device in a low-power-standby mode and typically draws 10 microamperes ( $\mu A$ )—a factor of 50 reduction over NMOS!

Refresh current, the third term in the equation, is cycle-time dependent. Current increases with the frequency of refresh. In dynamic RAMs, data is stored on a capacitor that must be replenished or recharged every 2 or 4 milliseconds (ms). This refresh time is a function of the stored charge and the leakage current. With the CMOS dynamic RAM, the cell storage capacitance is 0.125 picofarad (pF) compared to 0.040 pF to 0.085 pF in an NMOS dynamic RAM. This low capacitance, coupled with lower leakage currents, permits the CMOS refresh period to be extended to 64 ms in standby.

At the standard 128 refresh cycles/2 ms (equivalent to a 15.625- $\mu s$  refresh period), the NMOS device draws about 4.8 milliamperes (mA) and

asymptotically approaches the standby current of 4 mA as the refresh period approaches infinity. Even eliminating refresh entirely only reduces the current to 4 mA, which is only a 16 percent improvement. As a result, extending NMOS refresh does not significantly reduce the system's power consumption.

Contrast this characteristic to the improvement CMOS offers. At 15.625  $\mu s$ , the CMOS dynamic RAM draws approximately 10 percent of the NMOS current, or 0.42 mA at TTL levels. Extending the refresh period reduces the current asymptotically to the standby current of 0.05 mA. At a 64-ms refresh period, the current is reduced to 0.15 mA, a 300 percent reduction. When battery powered, the CMOS system has a 10 times longer life than does the NMOS system, and an extended refresh mode offers another fivefold improvement. A 256K-byte CMOS memory can retain data for nearly one week on only AA nickel-cadmium (nicad) cells—more than sufficient for most portable systems.

### High-Speed Applications

Ripplemode and Static Column mode are ideal for applications involving high-speed buffers, telecommunications, and graphics. Bit-mapped graphics systems would seem to be a natural fit with Page Mode operation. However, this was not always the case. Prior to the Intel 2164A 64K by 1-bit NMOS dynamic RAM, it was difficult to retrieve all 256 bits within a single row of memory because of the RAS-low time limitation of 10  $\mu s$ . Even with a Page Mode cycle time of 125 ns, to retrieve all 256 bits would require 32  $\mu s$ —three times longer than allowed. The 2164A extended the RAS-low time to 75  $\mu s$ , permitting the extraction of all 256 bits during a single Page Mode cycle.

At the end of the cycle, the device cannot be reaccessed again until after a certain off-time allows internal nodes to be precharged to be ready for the next cycle. As a result, the 2164A can stream data at greater than



a 7-MHz rate continuously. This function matches the timing and operation of low-performance, bit-mapped graphics memories. One 2164A, for example, can map all the data for the 256 by 256 matrix of a graphics display. During the horizontal scan time, the RAM performs a Page Mode cycle and one full line is displayed. During retrace time, the memory must be refreshed and can be updated with new data if required. This type of update is relatively slow; consequently, it limits the speed of animation on the screen because the processor has access to the memory only 25 percent of the time.

To increase resolution, more lines, each with more pixels, must be used. By performing two sequential Page Mode cycles from two different RAMs, pixel densities to 512 bits per line can be achieved. As pixel density increases, the memory cycle time must decrease to paint more pixels on a line in the same amount of time. This cycle-time limitation plus the fact that memory can be updated only during blanking has precluded dynamic RAMs from use in higher-resolution graphics displays. These systems are usually built with high-speed, expensive static RAMs.

With Riplemode, memory update during screen display time, also known as cycle stealing, is possible. As an example, a 512 by 384 display requires 512 bits/line and 1 bit every 67 ns. Data is read from four memory devices in a series of eight Riplemode reads each. Data is temporarily stored in a video-output register file and then shifted to the video screen at a rate slower than the Riplemode reads. Following this, enough time is available to perform an update cycle before the next eight Riplemode reads are performed to continue screen refresh. Eight was the number chosen to minimize the time the processor must wait to update the memory. In addition to this cycle stealing, which updates during display time, memory updates are also performed during blanking. Along with this system, a similar system was built using 2164As with Extended Page Mode operation. Each system used an iAPX 86 processor

and similar software. A comparison of both systems showed the CHMOS (complementary high-speed metal-oxide semiconductor) system to have a 42 percent higher drawing speed. Animation on the CHMOS system was vastly improved.

### Usable Speed

Memory design using dynamic RAMs has always been a challenge. Although multiplexing addresses does reduce the package pin count and increase system density, it limits the access and cycle times in the system. To access a dynamic RAM, low-order row addresses are presented and latched into the dynamic RAM with RAS. Row addresses must be held for a period  $t_{RAH}$  after the fall of RAS to guarantee proper operation. Next, the addresses must be changed to high-order column addresses and latched into the dynamic RAM with CAS, creating a timing window  $t_{RCD}$ , which is the RAS-to-CAS delay.

Within this window, the designer must guarantee row address hold time, change the addresses, and account for any timing skew on the CAS signal. If column addresses are valid at the maximum specified  $t_{RCD}$ , access time  $t_{RAC}$  is measured from the high-to-low transition of RAS.

The cycle time is the sum of the access time and the cycle precharge time  $t_{RP}$ . The access time is a function of  $t_{RCD}$ , which has contradictory requirements. It must be as long as possible to simplify system design and at the same time as short as possible to enhance system speed. Cycle time is affected directly by the length of  $t_{RP}$ .

Static-column operation eliminates the  $t_{RCD}$  problem. After row addresses have been latched into the RAM, the second portion of the access begins from valid column addresses. In other words, column access does not wait for CAS to become valid, but operates in a fashion similar to that of a static RAM. This is due to the flow-through operation of the CAS latch. CAS serves only to latch the addresses and to provide an output enable. Access from valid column addresses simplifies design by remov-

ing the CAS signal from the critical timing path.

Systems using dynamic RAMs are typically CAS access-limited because controllers generate timing signals in discrete clock increments. A CMOS dynamic RAM system might operate at 8 MHz without Wait states. Using any other 64K-bit dynamic RAM would require the injection of one or two Wait states, resulting in a corresponding performance penalty. Consequently, the advantage of higher processor speed is negated without the high-speed dynamic RAM. For systems incorporating either discrete or LSI controllers, the CMOS dynamic RAM simplifies the system design and offers higher system performance.

### High Reliability

Soft errors are random, nonrecurring failures caused by ionizing radiation present within the environment. All matter contains small amounts of radioactive material. Alpha particles emitted by an IC's packaging material can penetrate the enclosed circuit. As they do so, they generate hole-electron pairs. Any high-impedance node in the vicinity sensitive to 1 million electrons may be affected, because the difference between a 1 and a 0 (known as the critical charge) is about 1 million electrons. Consequently, data in one cell could change from a 1 to a 0 or vice versa. Correct data can be rewritten into the affected cell and the memory will again function correctly, thus the term "soft error."

When first discovered during tests of 16K-bit dynamic RAMs, soft errors occurred at a rate five times greater than catastrophic or hard-error failures. While device designers worked to eliminate the alpha-particle sensitivity, systems designers added error-correcting circuits (ECC), which increased system reliability, but the systems were larger and more expensive due to the additional components required. Also, the system had to test and correct the data, slowing the system's performance. All this was due to soft errors. Obviously, what is really required is the elimination of soft errors.



CMOS technology offers such a solution. The CMOS dynamic RAM cell is built on an n-well in a p-substrate, creating a p-n junction or diode at the boundary. When alpha particles create hole-electron pairs in a CMOS device, something else occurs. First, the n-well is very shallow, and the majority of hole-electron pairs are created in the p-substrate. Holes cannot transfer across the reverse-biased p-n junction, which acts as a barrier to soft-error effects. Any electrons that do cross the junction are gathered at the +5-V node away from the storage cell. The probability that sufficient hole-electron pairs are created within the n-well that cell upset could occur is so low that the soft-error rate of CMOS dynamic RAMs is typically orders of magnitude below that of their NMOS counterparts.

High storage capacitance also plays a role in the reduction of soft errors. The number of stored charged electrons representing a 1 or a 0 is directly proportional to the storage capacitance. Higher capacitance equates

to more stored charge, which in turn increases the critical charge. The critical charge is the number of particles that differentiate a 1 from a 0. Increasing the critical charge beyond 1 million electrons significantly reduces the susceptibility to soft errors. This, in addition to the n-well mechanism, reduces the soft-error rate to much less than 0.001 percent per 1000 hours.

Studies were performed to compare reliability of systems with and without error correction for both NMOS and CMOS dynamic RAMs. The results show one surprise: at 256K bytes and below, the CMOS system *without* ECC is more reliable than the NMOS system *with* ECC, because of the cycle-time dependence of soft errors. In small systems, the memory is accessed more frequently, and the probability of a soft error is increased. With a soft-error rate at the very minimum 100 times less than NMOS, the CMOS dynamic RAM does not experience this effect.

Systems below 256K-byte capacities

benefit by the elimination of ECC circuits from a cost, performance, and simplicity-of-design standpoint. First, ECC increases the access time of the system by 50 ns to check and correct data. Assuming a 120-ns RAM access, ECC increases the access by 42 percent. Moreover, the penalty on cycle time is even greater, especially when you are writing a single byte into a 2-byte word. In this instance, data must be accessed and corrected, the new byte merged into the word, and check bits generated. Finally, the system must write the new data into memory. Added to this are any system-timing skews. As a result, a 200-ns cycle time stretches to a 335-ns system cycle time or an increase of 68 percent. Therefore, using a CMOS dynamic RAM not only improves system reliability but enhances system speed and simplicity of design. ■

Joe Altnether is technical marketing manager at Intel Corp. (2111 N.E. 25th Ave., Hillsboro, OR 97123).

When first discovered during tests of 16K-bit dynamic RAMs, soft errors occurred at a rate five times greater than catastrophic or hard-error failures. While device designers worked to eliminate the alpha-particle sensitivity, system designers added error-correcting circuits (ECC) which increased system reliability, but the systems were larger and more expensive due to the additional components required. Also, the system had to test and correct the data, slowing the system's performance. All this was due to soft errors. Obviously,

Static-column operation eliminates the bus problem. After row addresses have been latched into the RAM, the second portion of the access begins: row valid column address. In other words, column access does not wait for CAS to become valid, but operates in a fashion similar to that of a static RAM. This is due to the low-through operation of the CAS latch. CAS serves only to latch the addresses and to provide an output

dynamic RAMs from use in high-resolution graphics displays. These systems are usually built with high-speed, expensive static RAMs. With ripple-mode memory update during screen display time also known as cycle stealing is possible. As an example, a 512 by 512 display requires 512 bitlines and 1 bit every 16 bits is read from memory. The data is read from four memory devices in a series of eight Ripples. Data is temporary made reads each. Data is temporary stored in a video-output register and then shifted to the video screen at a rate slower than the Ripples. Following this, enough time is available to perform an update cycle before the next eight Ripples. Reads are performed in continuous screen refresh. Light was the number chosen to minimize the time the processor must wait to update the memory. In addition to this, cycle stealing, which updates during display time, memory updates are also performed during blanking. Along with the system, a similar system was built using 256KAs with the

# Modular approach to C-MOS technology tailors process to application

Despite the proliferation of applications, a few C-MOS process variations can address the functional requirements of many different products

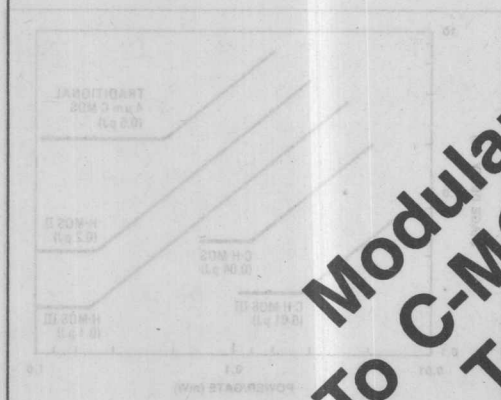
by Kim Kokkonen and Richard Pashley, Intel Corp., Santa Clara, Calif.

In the past few years, the interest in complementary MOS technology and its applications to new products has grown. Traditional arguments for C-MOS center on its low power consumption, its ability to be fabricated in a single process, and its simple circuit design. With the advent of very large-scale integration, these arguments are taking on new meaning and importance.

As an example, Fig. 1 compares the performance of H-MOS (high-performance n-channel MOS) inverters with their equivalent in Intel's C-MOS (complementary high-performance MOS) technology. Though H-MOS's speed continues to improve with further scaling, its delay-power product is more than an order of magnitude higher than a C-MOS implementation with identical physical parameters. In a VLSI part with 20,000 gates, C-MOS could mean the difference between 1 and 10 watts of power dissipation, which might save the expense and difficulty of a high-power cooling system or extend a portable system's battery life by a factor of 10.

That C-MOS is a better choice for a portable system is not the only reason why it is a better choice. In addition, the flexibility of C-MOS technology allows for a wide range of process variations. The number of process variations is limited by the number of integrated circuit functions that can be implemented in a single process. The number of process variations is limited by the number of integrated circuit functions that can be implemented in a single process.

Given the presence of parametric silicon controlled rectifiers within every C-MOS chip, a current pulse of sufficient magnitude either inside or outside the chip may cause a catastrophic latchup. Many schemes have been proposed to control latchup, ranging from carefully controlling the layout (which imposes no burden on the technology) to a buried layer (which significantly in-



1. Power dissipation is the continuous improvement of H-MOS (high-performance MOS) by scaling the delay-power product for C-MOS (complementary MOS) is more than an order of magnitude lower in the typical integrated circuit.

## Modular Approach To C-MOS Technology Tailors Process To Application

KIM KOKKONEN AND RICHARD PASHLEY  
Intel Corporation

# Modular approach to C-MOS technology tailors process to application

Despite the proliferation of applications, a few C-MOS process variations can address the functional requirements of many different products

by Kim Kokkonen and Richard Pashley, Intel Corp., Santa Clara, Calif.

□ In the past few years, the interest in complementary-MOS technology and its applications to new products has exploded. Traditional arguments for C-MOS center on its low power dissipation, the large noise margins of complementary logic, and its simple ratioless design. With the advent of very large-scale integration, these arguments are taking on new meaning and importance.

As an example, Fig. 1 compares the performance of H-MOS (high-performance n-channel MOS) inverters with their equivalent in Intel's C-H-MOS (complementary high-performance MOS) technology. Though H-MOS's speed continues to improve with further scaling, its delay-power product is more than an order of magnitude higher than a C-H-MOS implementation with identical n-channel transistors. In a VLSI part with 50,000 gates, C-H-MOS could mean the difference between 1 and 10 watts of power dissipation, which might save the expense and difficulty of a sophisticated cooling system or extend a portable system's operating time by a factor of 10.

That C-MOS performance is now on a par with n-MOS technology has also accelerated its popularity. In addition, the density of C-MOS circuitry has improved dramatically with advances in technology. Finally, the number of process alternatives has grown so large that almost any integrated-circuit design can be supported with available C-MOS technology.

Unfortunately, the wave of enthusiasm for C-MOS and the needs of different applications have multiplied the number of approaches that C-MOS developers are taking. Several major issues remain in VLSI C-MOS design—namely latchup and soft-error prevention, interconnections, and logic-design techniques. A building-block approach with a limited number of basic process modules can be used to create a close-knit family of technologies that squarely addresses these issues and simultaneously supports a wide range of applications.

## The basis for C-H-MOS

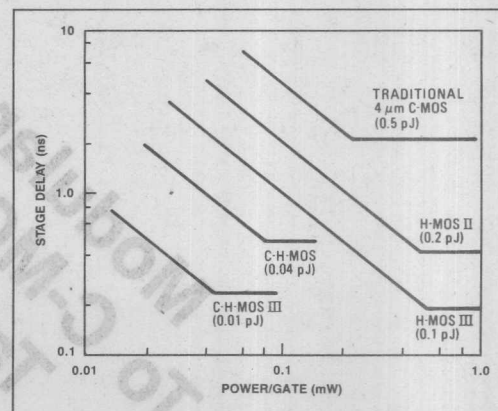
A firm foundation in n-MOS-transistor physics will support the advancement of C-MOS technology. As channel lengths approach 1 micrometer, n-channel transistors become more difficult to optimize because the standard 5-volt power supply causes problems with high-intensity fields. Improperly designed transistors may be unreliable as a result of hot-carrier injection into gate oxides, or they may cause less localized problems by injecting carri-

ers into the MOS substrate—there to bleed charge from storage nodes or even trigger a destructive latchup.

The resources to develop and verify the reliability of a 1- $\mu$ m n-channel transistor are well established and substantial. In Intel's C-H-MOS process, the basic design of the n-channel transistor is identical to its H-MOS counterpart, as shown in the table. Even at the more detailed levels of doping profiles, the H-MOS and C-H-MOS transistors are nearly identical.

Thus a high-performance C-MOS technology may be born out of an established n-MOS line. The relatively simple addition of an n-well in the same high-resistivity substrate results in a C-MOS process that serves as the basis for several optimized technologies. This is just a start, however, as other important issues remain.

Latchup has been the traditional nemesis of C-MOS. Given the presence of parasitic silicon controlled rectifiers within every bulk C-MOS chip, a current pulse of sufficient magnitude either inside or outside the chip may cause a catastrophic latchup. Many schemes have been proposed to combat latchup, ranging from carefully scrutinizing the layout (which imposes no burden on the technology) to a buried layer (which significantly in-



**1. Power down.** Despite the continuous improvement of H-MOS (high-performance MOS) by scaling, the delay-power product for C-H-MOS (complementary-MOS H-MOS) is more than an order of magnitude lower in the typical integrated circuit.

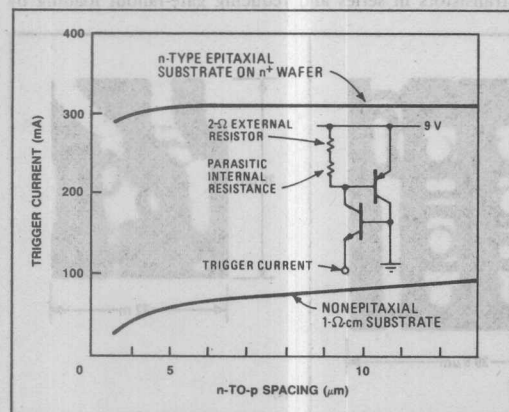
n-CHANNEL TRANSISTOR COMPARISON					
Technology		Gate-oxide thickness (Å)	Channel length (μm)	Threshold voltage (V)	Graded drain profile
n-MOS	C-MOS				
H-MOS I	—	700	3.0	0.7	no
H-MOS II	C-H-MOS	400	2.0	0.7	no
H-MOS II	C-H-MOS III	250	1.0	0.7	yes

creases complexity and processing cost). All have some degree of effectiveness.

A modular approach to a broad-based line of C-MOS technologies requires other measures besides mere physical latchup resistance. The latchup spoiler must be applicable to dynamic random-access memories, erasable programmable read-only memories, and static RAMs, as well as to microprocessors and controllers. In order to improve latchup resistance, it cannot increase the distance between n- and p-channel transistors (this constraint is most significant in random logic and full C-MOS six-transistor static-RAM cells). The technique must be compatible with low-cost and large-volume manufacturing. Finally, the approach must be consistent with the use of an automated checking algorithm, so that every gate of a large semirandom logic design need not be scrutinized for latchup susceptibility.

### Epitaxial benefits

Figure 2 shows the margin gained in latchup trigger current when an epitaxial substrate is used. The epitaxial substrate brings the same latchup benefits to all product lines, and in many cases provides additional advantages such as improved surface lifetimes (for dynamic RAMs) and reduced dc resistance (for E-PROMs and logic). Epitaxial substrates are now available in volume from commercial silicon vendors, adding less than 5% to the cost of a finished wafer. No additional or exotic fabrication equipment needs to be installed. Because the epitaxial



**2. Benefits.** By raising the margin of latchup trigger current, an epitaxial substrate effects a dramatic improvement in combating latchup, a major concern in complementary-MOS chip design.

substrate's heavily doped bulk effectively eliminates the vertically triggered latchup mode, it is possible to develop a set of computer-aided-design tools that can flag latchup-sensitive layouts on the largest VLSI chips.

Since grasping the phenomenon of upsets induced by alpha particles, in 1977, memory designers have taken care to ensure that enough charge is stored within each cell to minimize

the problem. As critical chip dimensions are reduced, this problem becomes more severe, however, since both parasitic and storage capacitances are naturally reduced. For the latest 1.5-μm n-MOS process, stored charge is low enough to caution even microprocessor designers to guard against random storage nodes suffering from soft errors. Fortunately, C-MOS provides a natural barrier against soft errors if the storage node is located within the C-MOS well.

The well junction is reverse-biased by the power-supply voltage. The electric field at this junction naturally repels any carriers generated outside the well that might otherwise diffuse up to surface storage nodes. The combination of the well structure and an epitaxial substrate is even more effective. Here the funneling mechanism that usually collapses local electric fields during the passage of an alpha particle is also minimized. By using epitaxial substrates and the protection of a C-MOS well, the amount of charge collected during an alpha event can be reduced by an order of magnitude.

Of course, the designer must arrange for the storage node to reside within the well. This constraint, combined with other performance issues, leads to different choices of well and substrate polarities, depending on application. For example, in a C-MOS technology that is optimized for dynamic RAM, the ideal memory cell should have a p-channel pass gate and a p-channel capacitor located within an n-well in a p-type substrate. The p-channel transistor is chosen because it injects far fewer spurious carriers into the substrate and thus does not by itself disturb the state of neighboring cells.

The conductance of the p-channel device, while lower than that of an n-channel device of the same size, does not degrade the RAM's performance, since dynamic-RAM sensing is limited primarily by the amount of stored charge. Experimental results with C-H-MOS dynamic RAMs based on these principles show a soft-error rate of less than 300 FIT (failures in time, or device failures per billion hours) at a power supply of only 3 V. This is an improvement of more than three orders of magnitude over traditional n-MOS dynamic-RAM technology and offers the possibility of dynamic-RAM systems that require no error correction and that are compatible with low-voltage battery backup.

High-density, high-performance static RAMs present the other side of the coin. The smallest static-RAM cells today are built using polysilicon-load resistors that sustain the stored-node voltage. On the time scale of an alpha event, however, these resistors in effect do not exist. Because the storage node's RC time constant is on the order of milliseconds and the alpha event's time scale



is nanoseconds, the cell appears dynamic. In this sense, polysilicon-load static-RAM cells are very similar to dynamic-RAM cells. The major difference arises in the way the cells sense the cell's information. The static-RAM cell provides a direct current, and to maximize the cell's performance, that current must be as large as possible while contained in a minimum area. Thus the chip designer must use high-gain n-channel transistors for the cell's pass gates and pulldowns. For good soft-error protection, then, the cell must be located in a p-well within an n-type substrate.

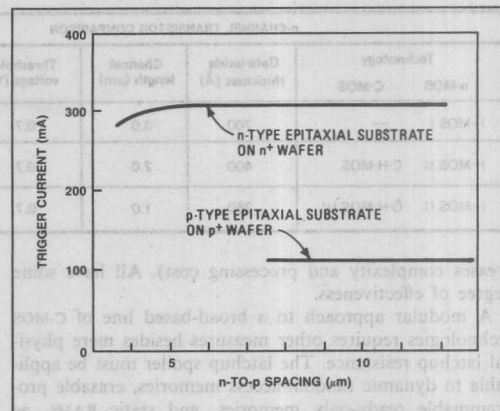
The p-well approach benefits even full C-MOS six-transistor static-RAM cells. The area of such cells depends strongly on the distance allowed between n- and p-channel devices. Using a straightforward implementation of epitaxial C-MOS, the p-well approach provides more margin against latchup at small n-to-p spacings (Fig. 3). This phenomenon occurs because of the differing diffusion properties of n- and p-type dopants. The heavy doping in the n-type substrate is less mobile than is the p-type dopant, resulting in less outdiffusion during thermal processing and thus minimizing the shunt resistance that controls latchup.

### Hooking it up

One of the challenges of C-MOS in logic applications is interconnection. Designers of n-MOS chips are accustomed to buried contacts, which directly connect n-type polysilicon and n-type transistor source or drain regions. Because C-MOS requires contact to both p and n regions, the traditional n-type buried contact becomes much less useful, and a version suitable for both diffusion polarities is quite difficult to implement. This increases the burden on contact and metallization modules.

For high-density C-MOS logic, the first level of metal is all but consumed by local connections between p and n transistors. The payback from adding a second level of metal for longer-distance routing is very high. A good example exists for the six-transistor static-RAM cell commonly used by logic designers. Figure 4 compares single- and double-metal versions of this cell, both implemented with 1.5- $\mu\text{m}$  design rules. Here the second-metal layer provides the bit lines for the cell. Similar arguments justify the use of second metal in global power, clock, and data routing in complex microprocessor chips.

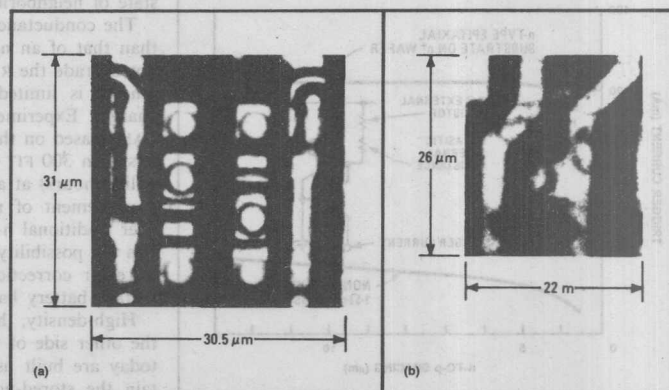
Contacts themselves are more difficult to build in C-MOS. N-MOS technology accustomed process engineers to adding a phosphorus contact plug after the contacts have been etched. This plug brought several advantages: the phosphorus gettering metallic contaminants from the wafer, reducing junction leakage; and the high-temperature diffusion rounded the profile of the contact sidewall, easing the step coverage of the metal subsequently deposited. Further, the plug had self-aligning features. If the



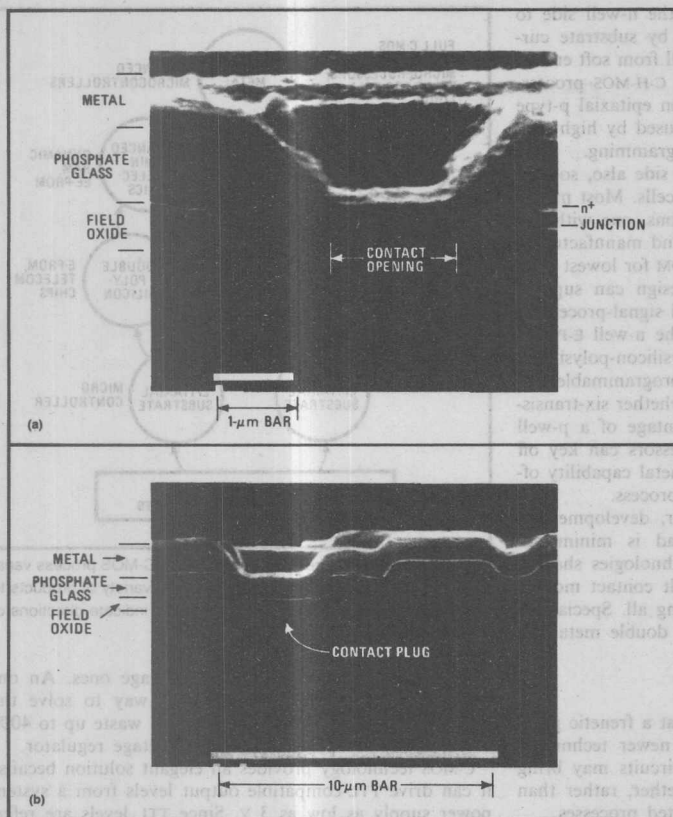
**3. P-well margin.** With an epitaxial substrate, a p-well structure (upper curve) yields a greater margin against latchup than n-well at smaller n- and p-device spacings.

contact etch attacked the silicon substrate or if the contact was misaligned toward the field-oxide edge, the plug would rejuvenate the resulting weakened junctions. In C-MOS, these same attributes must be obtained differently, through improved fabrication, cleanliness, new gettering techniques, improved dielectrics, and tightly controlled contact etching. Figure 5 shows the difference in implementing a 1.5- $\mu\text{m}$  contact structure in n-MOS and C-MOS.

Along with the proliferation of C-MOS technologies has come a wave of innovation in C-MOS design techniques. For digital logic, the major contenders for broad use are full complementary design and domino logic, first proposed by AT&T Bell Laboratories (Fig. 6). For many applications, traditional C-MOS logic is a winner. It requires no clocks, has larger operating margins, and uses fewer transistors for simple gates. For more complex gates, however, domino logic uses fewer transistors and runs faster. The speed results from connecting fewer transistors in series and reducing gate-fanout loading by



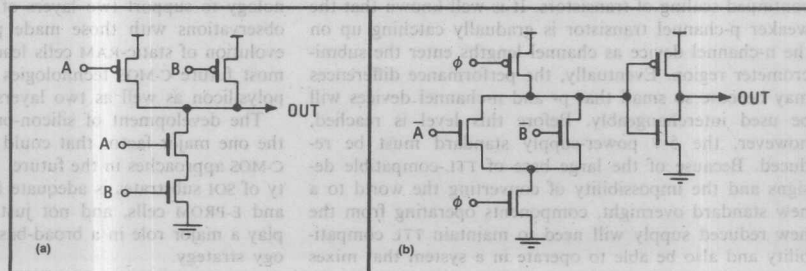
**4. Payback.** The use of double-metal layers for a six-transistor static-RAM cell can produce a large savings in real-estate. In two cells implemented with a 1.5- $\mu\text{m}$  design rules, the savings can amount to one third of the total area. The cell at right uses second-layer metal for bit lines.



up to a factor of two compared with full C-MOS.

Interestingly, the choice of design style influences the optimal type of C-MOS well. The speed of full C-MOS is limited by the slower of the two transistor polarities. Since the trip point is quite close to half the power supply, the time required for either transistor type to discharge its load capacitance by about 2.5 V sets the gate's speed. Since the p-channel device is the weaker one, it pays to choose a well type that improves the p-channel's conductance. P-well does this because the p-device is fabricated in an uncompensated substrate and thus has maximum mobility. Comparisons between n-

**6. Logic.** Two major contenders for digital logic design are full complementary (a) and domino logic (b). The former requires no clocks and is simpler for many applications. Domino logic, which performs best in an n-well technology, is faster and simpler for more complex circuits.



**5. Making contact.** Contacts are more difficult to build in C-MOS than in conventional n-MOS. The phosphorous contact plug used in n-MOS after contact etching (a) adds desirable features such as reduced junction leakage and improved step coverage by the metal layer. To gain the same advantages in C-MOS requires greater process control (b).

and p-well construction show that the p-channel's conductance may be improved by as much as 10% with the proper well type.

By contrast, domino logic is at its best in an n-well technology. Here, the n-channel transistor dominates both performance and transistor count. Placing the n-channel device outside the well improves its conductance and reduces the dominant parasitic junction capacitance. Density also increases because no well contacts are required for the majority of the transistors.

The twin-well approach to C-MOS blurs these distinctions. In this approach, a high-resistivity epitaxial layer is grown on a heavily doped starting wafer. Then the doping for each transistor polarity may be independently optimized without need for doping compensation. Performance arguments based on mobility or junction capacitance thus become moot. Nonetheless, domino logic will still be best on a p-type substrate (equivalent to n-well) because it does not require well contacts to collect the large parasitic substrate currents from the n-channel transistors, thus improving packing density.

### Matching process to product

These and other technical arguments may be combined into a consistent strategy (Fig. 7) for creating a line of C-MOS processes serving a broad marketplace. For at least the next several years, a complete technology line must include C-MOS based on both p- and n-type substrates. Fortunately, choosing epitaxial-latchup control minimizes the development cost of running both process-

es. Dynamic RAMs are supported on the n-well side to minimize pattern sensitivities induced by substrate currents while protecting the p-channel cell from soft errors. E-PROMs are built in a similar n-well C-H-MOS process. Placing Intel's n-MOS E-PROM cell in an epitaxial p-type substrate eliminates parasitic effects caused by high substrate currents flowing during cell programming.

Microcontrollers land on the n-well side also, so that they may incorporate on-chip E-PROM cells. Most microcontroller products come in two versions, one with on-chip E-PROM for system-development and manufacturing flexibility, and another with on-chip ROM for lowest cost. Using n-well C-MOS, a single core design can support both versions. Telecommunications and signal-processing products can also take advantage of the n-well E-PROM process, both for its high-quality polysilicon-polysilicon capacitors and for the E-PROM cell's programmable features. High-performance static RAMs, whether six-transistor or polysilicon-load, can take advantage of a p-well C-H-MOS process. High-end microprocessors can key off the dense n-to-p packing and double-metal capability offered by the six-transistor static-RAM process.

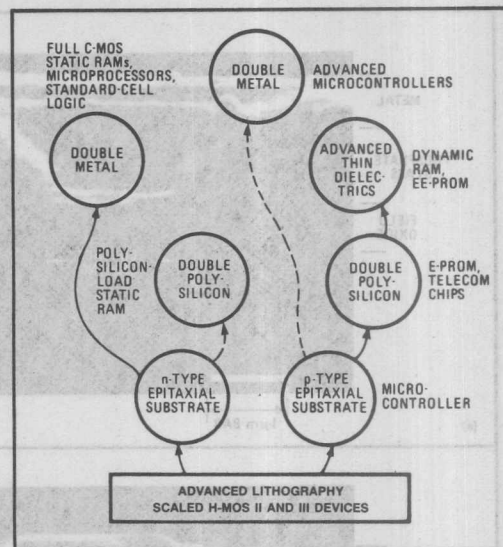
Because these processes are modular, development is simplified and manufacturing overhead is minimized. Just as all the 1.5- $\mu$ m C-H-MOS III technologies share a common transistor module, the difficult contact module was developed once to be shared among all. Specialized features such as double polysilicon or double metal are extensions of the common base.

### The future

C-MOS technology is still developing at a frenetic pace. Surprisingly, the application of some newer techniques and the demands of next-generation circuits may bring the various forms of C-MOS closer together, rather than further splitting the number of integrated processes.

One example of this trend is the development of a trench-isolation technique for separating n and p devices. When this module is perfected, there will be no reason to develop six-transistor static-RAM cells on p-well technology. The near-ideal trench isolation will prevent latchup on either substrate type. Similarly, if stacked C-MOS static-RAM cells can be perfected, there will be no need for polysilicon loads. The stacked C-MOS cell will have the same density but with improved performance and soft-error immunity. At that time, twin-well C-MOS on a p-type substrate, augmented by specialized features for specific product lines, will become the one approach to a broad line of C-MOS processes.

Another factor affecting future C-MOS integration is the continued scaling of transistors. It is well known that the weaker p-channel transistor is gradually catching up on the n-channel device as channel lengths enter the submicrometer region. Eventually, the performance differences may become so small that p- and n-channel devices will be used interchangeably. Before this level is reached, however, the 5-v power-supply standard must be reduced. Because of the large base of TTL-compatible designs and the impossibility of converting the world to a new standard overnight, components operating from the new reduced supply will need to maintain TTL compatibility and also be able to operate in a system that mixes



**7. Technology tree.** A relatively small line of C-MOS process variations, or modules, can be matched to a wide variety of products to serve a broad marketplace. The broken lines indicate directions of potential future growth.

older 5-v components with lower-voltage ones. An on-chip 5-to-3-v converter may be one way to solve the problem. This technique, however, will waste up to 40% of the total chip power within the voltage regulator.

C-MOS technology provides an elegant solution because it can drive TTL-compatible output levels from a system power supply as low as 3 v. Since TTL levels are referenced to the negative (ground) rail, the grounded substrate offered by n-well C-MOS is a much-preferred means of integrating submicrometer transistors into such a system. This will be a strong motive to standardize on p-substrate C-MOS.

A final factor that tends to drive future C-MOS processes toward commonality is the growing importance of RC delays in overall chip performance. The latest high-performance static RAMs use an aluminum strap in parallel with the polysilicon word line because the RC delay induced by even the best refractory metal polycides is several nanoseconds too long. Studies of dynamic RAMs larger than 1 megabit similarly indicate that refractory word lines will probably be inadequate, forcing the technology to support two layers of metal. Combining these observations with those made previously regarding the evolution of static-RAM cells leads to the conclusion that most future C-MOS technologies will have two layers of polysilicon as well as two layers of metal.

The development of silicon-on-insulator technology is the one major factor that could renew the divergence of C-MOS approaches in the future. However, until the quality of SOI substrates is adequate to support dynamic RAM and E-PROM cells, and not just static logic, it will not play a major role in a broad-based and modular technology strategy. □

---

Advanced  
Packaging Information

---

24





# Packaging Information

## CHAPTER 24 ADVANCED PACKAGING

### 24.1 Introduction

Today, system designers using LSI and VLSI devices are continuously facing problems associated with achieving the highest system performance level and most complex functional level of a particular system application in the smallest physical size possible. Until recently the available solutions to these device problems were limited to the traditional standard dual-inline-package (DIP) based on 100mil center-to-center lead spacing and flat packages (FP). Today, these device problems are being solved in a number of new ways; DIPs based on 50 mil center-to-center lead spacing, surface mounted small outline DIP, surface mounted chip carriers, surface mounted gull-wing flat package and pin grid arrays. Among these possible solutions the two that are emerging as the next standard IC packages for LSI/VLSI devices are the surface mounted chip carrier and the pin grid array.

### 24.2 What Are Surface Mounted Chip Carriers?

The chip carrier is basically the business portion of a DIP. Chip carriers are available in two general types: leadless and leaded.

The leadless chip carrier construction is accomplished in much the same manner as the multi-layer ceramic DIP package, but it is missing the side-brazed legs and much of the ceramic surrounding the die cavity area. Instead, it consists of a ceramic package with I/O pads on all four sides, a die cavity area, metalization traces to the I/O pads, and a hermetically sealable lid. Leadless chip carriers are available in either square or rectangular package outlines. The leadless chip carrier can be attached to a board surface either directly, by socketing, or by the addition of add-on leads.

The leaded chip carrier construction is accomplished also in much the same manner as the plastic DIP package, but has leads that are bent down and under the package on all four sides rather than like the DIP. It is also missing much of the plastic surrounding the die platform and consists of plastic material encapsulating the die platform with I/O pads or leads on all four sides. The plastic leaded chip carrier is available in square or rectangular package outline, that can be attached to a board surface either directly or by a socket.

Chip carriers are registered JEDEC standard packages. The standard is based upon two basic package types, one with 50mil center-to-center terminal spacing, the other with 40mil spacing. Each package type was developed

with certain application in mind, such as mounting methods, board material, thermal characteristics and external features. Intel's Microcontroller Operation offers two varieties of JEDEC packages; both with 50mil spacing, a square ceramic leadless type and a square plastic leaded type.

In addition to these mechanical packaging advantages, there are also electrical benefits. The package arrangement of the chip carriers I/O pads, on all four sides, allows for package traces to be shorter and more uniform in length. This allows lower resistance, less capacitance and less inductance, resulting in higher system performance and improved switching characteristics.

### 24.3 Why Chip Carriers?

Figure 24.1 shows the differences between the surface area ( $\text{in}^2$ ) versus pin count of both a ceramic dip and a ceramic leadless chip carrier device. Note that an 18-pad chip carrier offers a 57% saving in area. As the pin counts increase, the chip carrier surface area advantage becomes significantly more obvious. This space efficiency allows the system designer to increase the number of components on a board or decrease the overall board size and, thus, the overall system size.

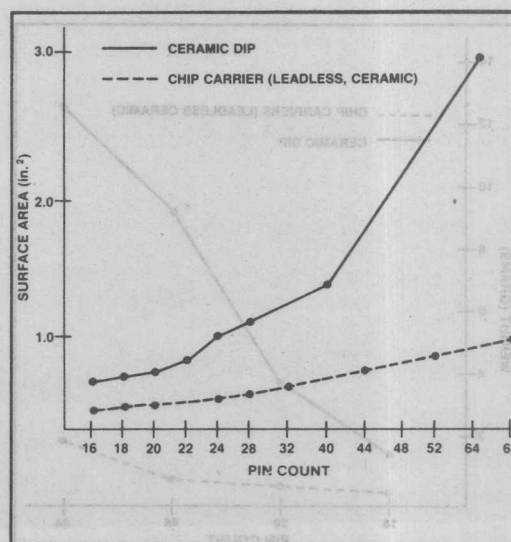


Figure 24.1. Package Area

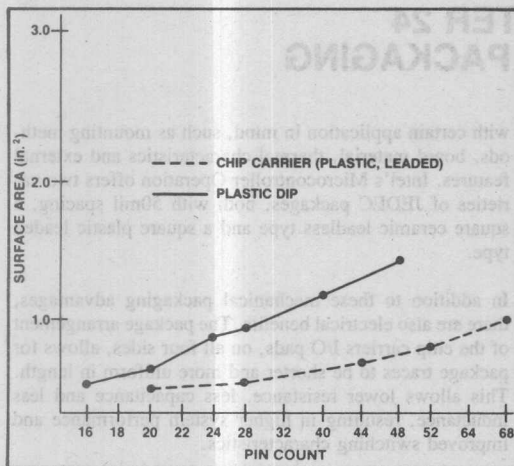


Figure 24.2. Package Area

Figure 24.2 shows the differences between the surface area (in<sup>2</sup>) versus pin count of both a plastic DIP and a plastic Leaded Chip Carrier device. As can be seen, the savings in area is also as significant as the pin count increases, allowing the same system benefits.

However, the biggest advantage a ceramic chip carrier has over a ceramic DIP is in its weight. Figure 24.3 shows the difference between the weight (grams) versus pin count of both ceramic DIP and ceramic Leadless Chip Carrier.

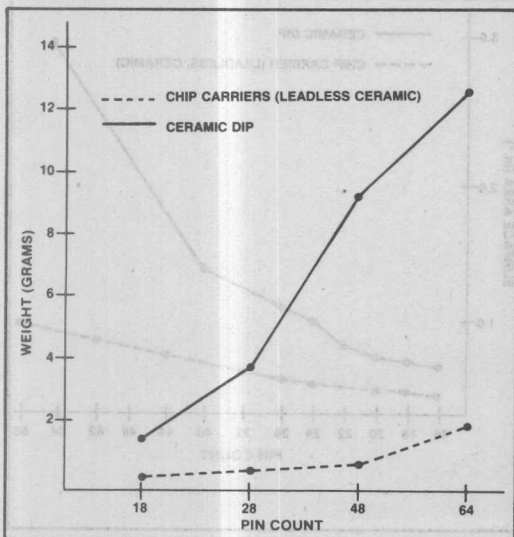


Figure 24.3. Package Weight

For an 18-pin count, there is a 77% weight saving and as the pin counts increase to 28 and to 48, this weight savings increases to 90% and 95%, respectively. In the case of the plastic leaded chip carrier, the weight savings over a plastic DIP is noticeable but not significant. However, it is in the ability to decrease the board size and, thus, economizing on material and weight reduction that the significant advantage exists.

#### 24.4 What Are Pin Grid Arrays?

The Pin Grid Array is basically a combination of the ceramic DIP and ceramic Leadless Chip Carrier. The Pin Grid Array construction is accomplished in much the same manner as the multi-layer ceramic DIP, but it is missing the side-brazed legs and much of the ceramic surrounding the die cavity. Instead, it consists of a ceramic package with leads coming off the bottom in rows or circular patterns. The Pin Grid Array is available in square package outlines with lead spacing of 100 mils and can be attached to a board in the same manner as a DIP.

Pin Grid Arrays are being proposed as JEDEC standard packages and will have from 1 to 10 nested rows of legs and may have a die cavity mounting area oriented up or down.

#### 24.5 Why Pin Grid Array?

Figure 24.4 shows the difference between the surface area (in<sup>2</sup>) versus pin count of both a 50 mil spacing chip carrier and a 100 mil spacing Pin Grid Array device. Note that at approximately 68 pins and above the Pin Grid Array becomes a better solution for higher pin count requirements than the chip carrier. In addition, the 100 mil lead spacing and through board mounting technique provides customers with an assembly technology that is familiar.

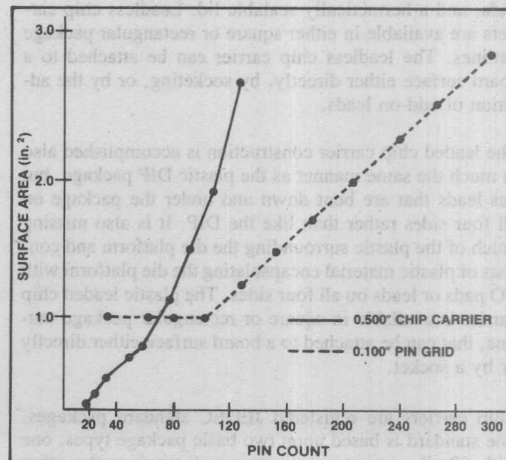
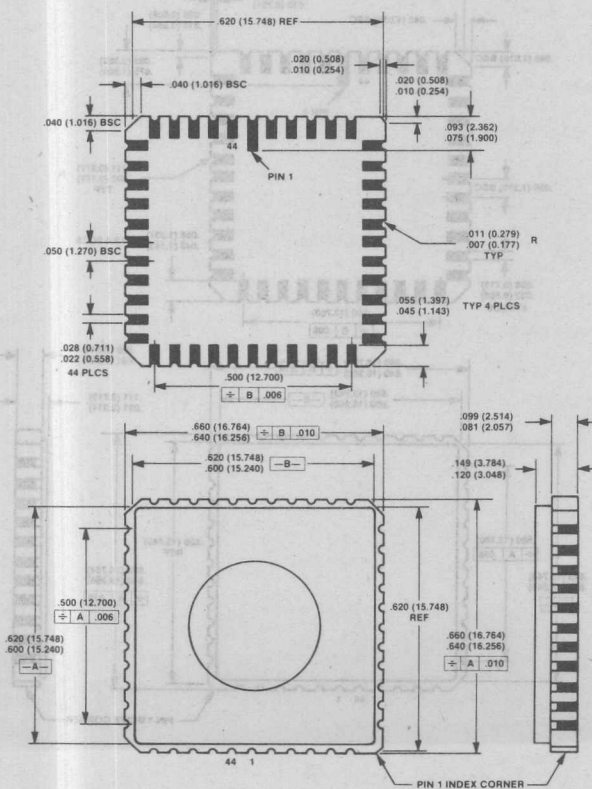


Figure 24.4. Surface Array



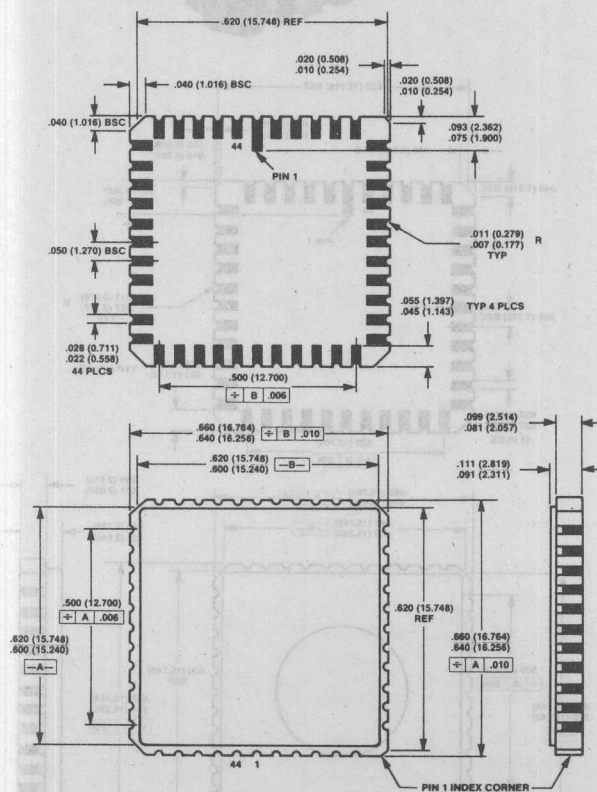
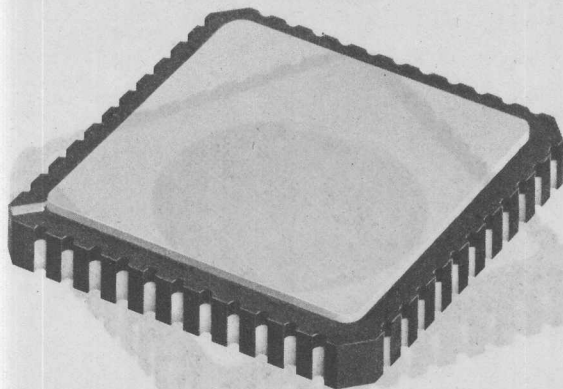


## ADVANCED PACKAGING

### CERAMIC LEADLESS CHIP CARRIER

All dimensions in inches and (millimeters)

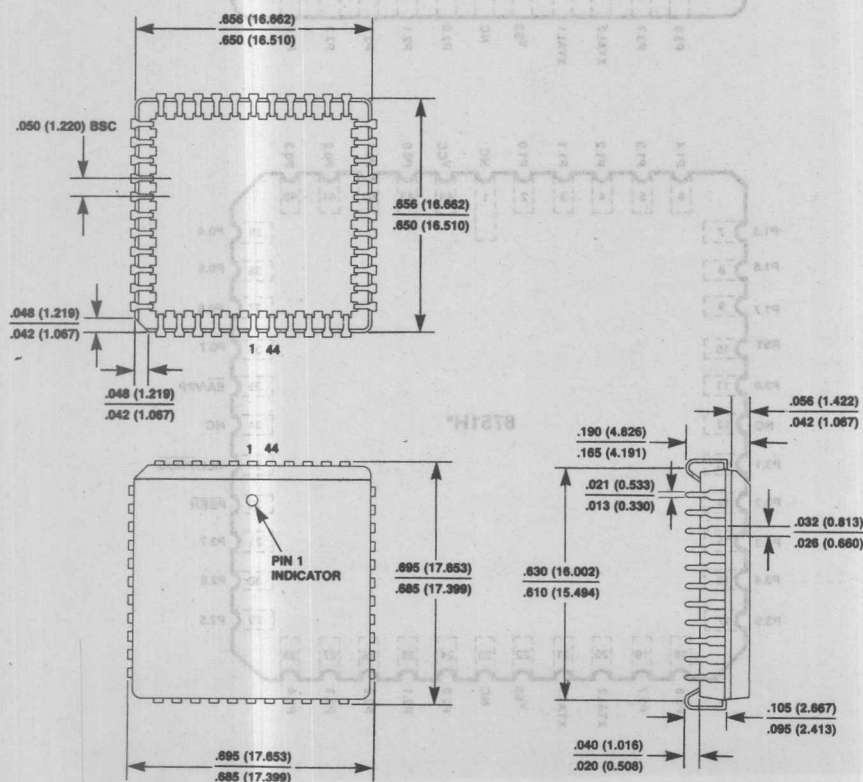
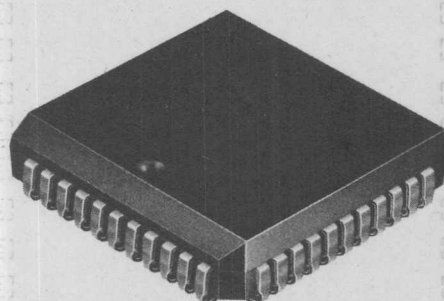
#### 44-Leadless Hermetic Chip Carrier JEDEC Package Type C



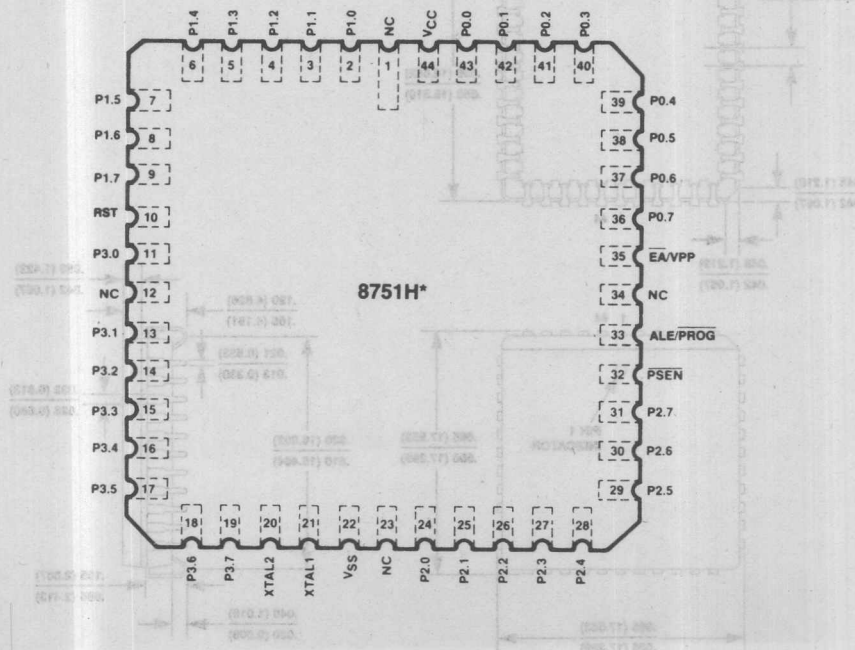
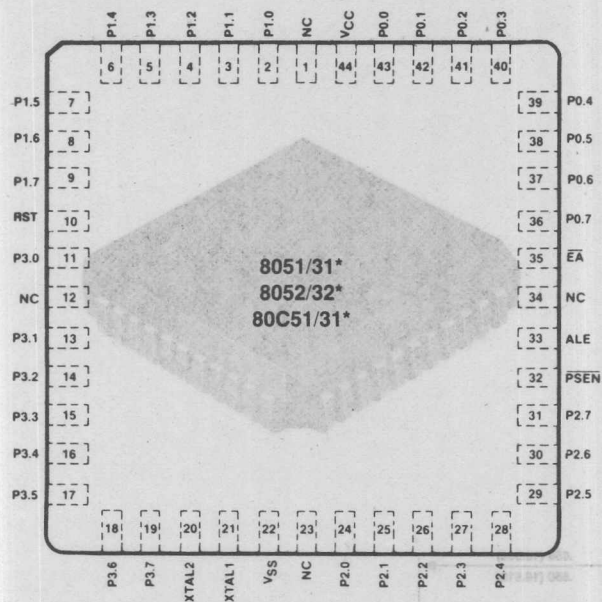
# PLASTIC LEADED CHIP CARRIER

All dimensions in inches and (millimeters)

44-Leaded Chip Carrier  
JEDEC Package Type C

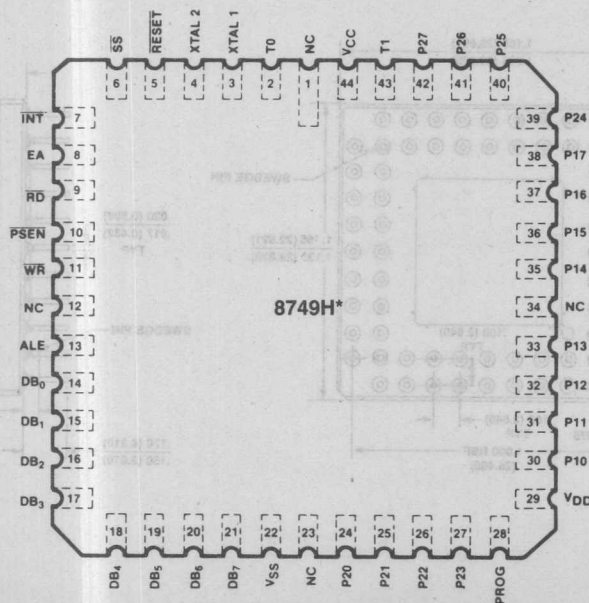
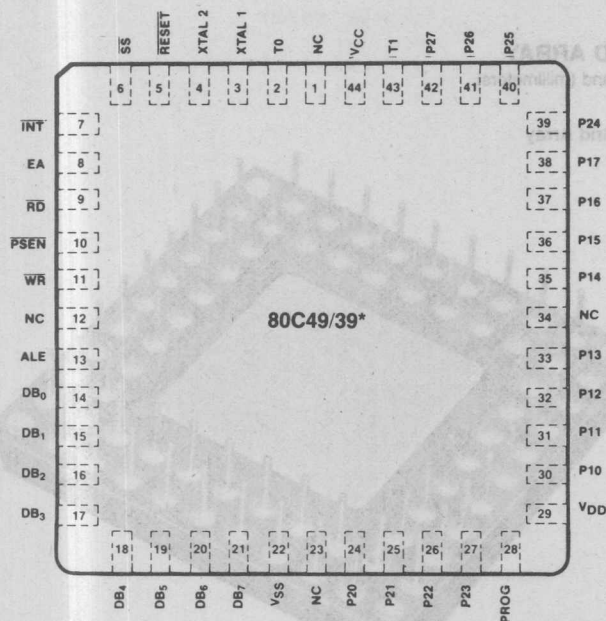


Pinouts  
MCS<sup>®</sup>-51 Family



\*Top View Looking Through Package

Pinouts  
MCS<sup>®</sup>-48 Family

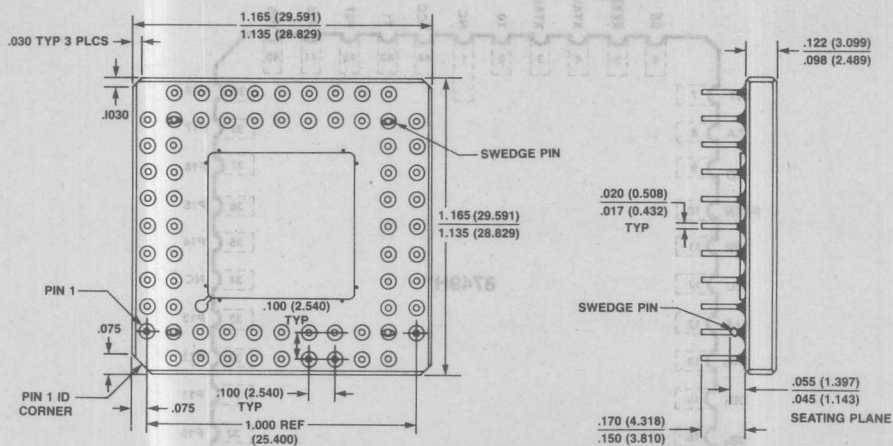
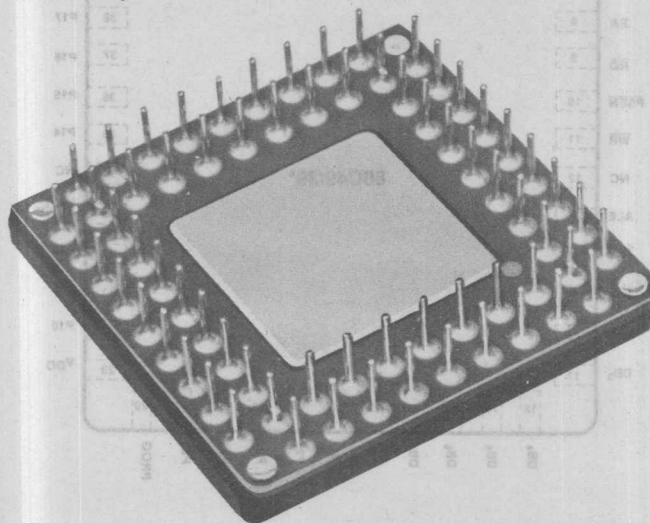


\*Top View Looking Through Package

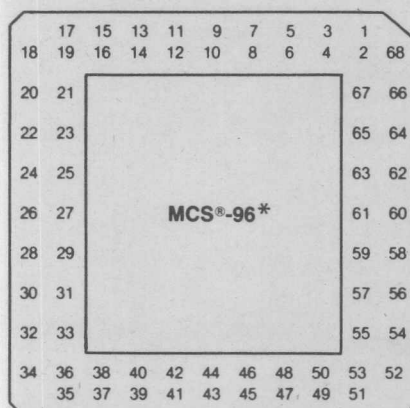


All dimensions in inches and (millimeters)

### 68-Pin Hermetic Pin Grid Array



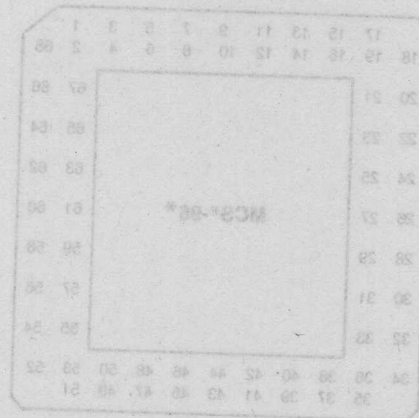
Pinouts  
MCS®-96 Family



MCS®-96 Pin Table

PIN	SYMBOL	PIN	SYMBOL	PIN	SYMBOL	PIN	SYMBOL
1	ACH7/P0.7	18	AD0/P3.0	35	READY	52	HSI2/HSO4
2	ACH6/P0.6	19	AD1/P3.1	36	T2RST/P2.4	53	HSI1
3	ACH2/P0.2	20	AD2/P3.2	37	BHE	54	HSI0
4	ACH0/P0.0	21	AD3/P3.3	38	WR	55	P1.4
5	ACH1/P0.1	22	AD4/P3.4	39	PWM/P2.5	56	P1.3
6	ACH3/P0.3	23	AD5/P3.5	40	P2.7	57	P1.2
7	NMI	24	AD6/P3.6	41	VBB	58	P1.1
8	EA	25	AD7/P3.7	42	VSS	59	P1.0
9	VCC	26	AD8/P4.0	43	HSO3	60	TXD/P2.0
10	VSS	27	AD9/P4.1	44	HSO2	61	RXD/P2.1
11	XTAL1	28	AD10/P4.2	45	P2.6	62	RESET
12	XTAL2	29	AD11/P4.3	46	P1.7	63	EXTINT/P2.2
13	CLKOUT	30	AD12/P4.4	47	P1.6	64	VPD
14	TEST	31	AD13/P4.5	48	P1.5	65	VREF
15	INST	32	AD14/P4.6	49	HSO1	66	ANGND
16	ALE	33	AD15/P4.7	50	HSO0	67	ACH4/P0.4
17	RD	34	T2CLK/P2.3	51	HSI3/HSO5	68	ACH5/P0.5

\* Top View Looking Through Package



MCS-86 Pin Table

PIN	SYMBOL	PIN	SYMBOL	PIN	SYMBOL	PIN	SYMBOL
1	ACHT.P0.7	33	AD0.P4.5	49	READY	65	AD0.P0.0
2	ACHT.P0.6	34	AD0.P4.4	50	TRST.P2.4	66	AD0.P0.1
3	ACHT.P0.5	35	AD0.P4.3	51	ONE	67	AD0.P0.2
4	ACHT.P0.4	36	AD0.P4.2	52	WR	68	AD0.P0.3
5	ACHT.P0.3	37	AD0.P4.1	53	PWM.P2.3	69	AD0.P0.4
6	ACHT.P0.2	38	AD0.P4.0	54	P2.7	70	AD0.P0.5
7	RAM	39	AD0.P3.9	55	V80	71	AD0.P0.6
8	EA	40	AD0.P3.8	56	V80	72	AD0.P0.7
9	VCC	41	AD0.P3.7	57	V80	73	AD0.P0.0
10	V80	42	AD0.P3.6	58	V80	74	AD0.P0.1
11	XTAL1	43	AD0.P3.5	59	V80	75	AD0.P0.2
12	XTAL2	44	AD0.P3.4	60	V80	76	AD0.P0.3
13	CLKOUT	45	AD0.P3.3	61	V80	77	AD0.P0.4
14	TEST	46	AD0.P3.2	62	V80	78	AD0.P0.5
15	TEST	47	AD0.P3.1	63	V80	79	AD0.P0.6
16	ALB	48	AD0.P3.0	64	V80	80	AD0.P0.7
17	RD	49	AD0.P2.9	65	V80	81	AD0.P0.0
		50	AD0.P2.8	66	V80	82	AD0.P0.1
		51	AD0.P2.7	67	V80	83	AD0.P0.2
		52	AD0.P2.6	68	V80	84	AD0.P0.3
		53	AD0.P2.5	69	V80	85	AD0.P0.4
		54	AD0.P2.4	70	V80	86	AD0.P0.5
		55	AD0.P2.3	71	V80	87	AD0.P0.6
		56	AD0.P2.2	72	V80	88	AD0.P0.7
		57	AD0.P2.1	73	V80	89	AD0.P0.0
		58	AD0.P2.0	74	V80	90	AD0.P0.1
		59	AD0.P1.9	75	V80	91	AD0.P0.2
		60	AD0.P1.8	76	V80	92	AD0.P0.3
		61	AD0.P1.7	77	V80	93	AD0.P0.4
		62	AD0.P1.6	78	V80	94	AD0.P0.5
		63	AD0.P1.5	79	V80	95	AD0.P0.6
		64	AD0.P1.4	80	V80	96	AD0.P0.7
		65	AD0.P1.3	81	V80	97	AD0.P0.0
		66	AD0.P1.2	82	V80	98	AD0.P0.1
		67	AD0.P1.1	83	V80	99	AD0.P0.2
		68	AD0.P1.0	84	V80	100	AD0.P0.3
		69	AD0.P0.9	85	V80	101	AD0.P0.4
		70	AD0.P0.8	86	V80	102	AD0.P0.5
		71	AD0.P0.7	87	V80	103	AD0.P0.6
		72	AD0.P0.6	88	V80	104	AD0.P0.7
		73	AD0.P0.5	89	V80	105	AD0.P0.0
		74	AD0.P0.4	90	V80	106	AD0.P0.1
		75	AD0.P0.3	91	V80	107	AD0.P0.2
		76	AD0.P0.2	92	V80	108	AD0.P0.3
		77	AD0.P0.1	93	V80	109	AD0.P0.4
		78	AD0.P0.0	94	V80	110	AD0.P0.5
		79	AD0.P0.0	95	V80	111	AD0.P0.6
		80	AD0.P0.1	96	V80	112	AD0.P0.7
		81	AD0.P0.2	97	V80	113	AD0.P0.0
		82	AD0.P0.3	98	V80	114	AD0.P0.1
		83	AD0.P0.4	99	V80	115	AD0.P0.2
		84	AD0.P0.5	100	V80	116	AD0.P0.3
		85	AD0.P0.6	101	V80	117	AD0.P0.4
		86	AD0.P0.7	102	V80	118	AD0.P0.5
		87	AD0.P0.0	103	V80	119	AD0.P0.6
		88	AD0.P0.1	104	V80	120	AD0.P0.7
		89	AD0.P0.2	105	V80	121	AD0.P0.0
		90	AD0.P0.3	106	V80	122	AD0.P0.1
		91	AD0.P0.4	107	V80	123	AD0.P0.2
		92	AD0.P0.5	108	V80	124	AD0.P0.3
		93	AD0.P0.6	109	V80	125	AD0.P0.4
		94	AD0.P0.7	110	V80	126	AD0.P0.5
		95	AD0.P0.0	111	V80	127	AD0.P0.6
		96	AD0.P0.1	112	V80	128	AD0.P0.7
		97	AD0.P0.2	113	V80	129	AD0.P0.0
		98	AD0.P0.3	114	V80	130	AD0.P0.1
		99	AD0.P0.4	115	V80	131	AD0.P0.2
		100	AD0.P0.5	116	V80	132	AD0.P0.3
		101	AD0.P0.6	117	V80	133	AD0.P0.4
		102	AD0.P0.7	118	V80	134	AD0.P0.5
		103	AD0.P0.0	119	V80	135	AD0.P0.6
		104	AD0.P0.1	120	V80	136	AD0.P0.7
		105	AD0.P0.2	121	V80	137	AD0.P0.0
		106	AD0.P0.3	122	V80	138	AD0.P0.1
		107	AD0.P0.4	123	V80	139	AD0.P0.2
		108	AD0.P0.5	124	V80	140	AD0.P0.3
		109	AD0.P0.6	125	V80	141	AD0.P0.4
		110	AD0.P0.7	126	V80	142	AD0.P0.5
		111	AD0.P0.0	127	V80	143	AD0.P0.6
		112	AD0.P0.1	128	V80	144	AD0.P0.7
		113	AD0.P0.2	129	V80	145	AD0.P0.0
		114	AD0.P0.3	130	V80	146	AD0.P0.1
		115	AD0.P0.4	131	V80	147	AD0.P0.2
		116	AD0.P0.5	132	V80	148	AD0.P0.3
		117	AD0.P0.6	133	V80	149	AD0.P0.4
		118	AD0.P0.7	134	V80	150	AD0.P0.5
		119	AD0.P0.0	135	V80	151	AD0.P0.6
		120	AD0.P0.1	136	V80	152	AD0.P0.7
		121	AD0.P0.2	137	V80	153	AD0.P0.0
		122	AD0.P0.3	138	V80	154	AD0.P0.1
		123	AD0.P0.4	139	V80	155	AD0.P0.2
		124	AD0.P0.5	140	V80	156	AD0.P0.3
		125	AD0.P0.6	141	V80	157	AD0.P0.4
		126	AD0.P0.7	142	V80	158	AD0.P0.5
		127	AD0.P0.0	143	V80	159	AD0.P0.6
		128	AD0.P0.1	144	V80	160	AD0.P0.7
		129	AD0.P0.2	145	V80	161	AD0.P0.0
		130	AD0.P0.3	146	V80	162	AD0.P0.1
		131	AD0.P0.4	147	V80	163	AD0.P0.2
		132	AD0.P0.5	148	V80	164	AD0.P0.3
		133	AD0.P0.6	149	V80	165	AD0.P0.4
		134	AD0.P0.7	150	V80	166	AD0.P0.5
		135	AD0.P0.0	151	V80	167	AD0.P0.6
		136	AD0.P0.1	152	V80	168	AD0.P0.7
		137	AD0.P0.2	153	V80	169	AD0.P0.0
		138	AD0.P0.3	154	V80	170	AD0.P0.1
		139	AD0.P0.4	155	V80	171	AD0.P0.2
		140	AD0.P0.5	156	V80	172	AD0.P0.3
		141	AD0.P0.6	157	V80	173	AD0.P0.4
		142	AD0.P0.7	158	V80	174	AD0.P0.5
		143	AD0.P0.0	159	V80	175	AD0.P0.6
		144	AD0.P0.1	160	V80	176	AD0.P0.7
		145	AD0.P0.2	161	V80	177	AD0.P0.0
		146	AD0.P0.3	162	V80	178	AD0.P0.1
		147	AD0.P0.4	163	V80	179	AD0.P0.2
		148	AD0.P0.5	164	V80	180	AD0.P0.3
		149	AD0.P0.6	165	V80	181	AD0.P0.4
		150	AD0.P0.7	166	V80	182	AD0.P0.5
		151	AD0.P0.0	167	V80	183	AD0.P0.6
		152	AD0.P0.1	168	V80	184	AD0.P0.7
		153	AD0.P0.2	169	V80	185	AD0.P0.0
		154	AD0.P0.3	170	V80	186	AD0.P0.1
		155	AD0.P0.4	171	V80	187	AD0.P0.2
		156	AD0.P0.5	172	V80	188	AD0.P0.3
		157	AD0.P0.6	173	V80	189	AD0.P0.4
		158	AD0.P0.7	174	V80	190	AD0.P0.5
		159	AD0.P0.0	175	V80	191	AD0.P0.6
		160	AD0.P0.1	176	V80	192	AD0.P0.7
		161	AD0.P0.2	177	V80	193	AD0.P0.0
		162	AD0.P0.3	178	V80	194	AD0.P0.1
		163	AD0.P0.4	179	V80	195	AD0.P0.2
		164	AD0.P0.5	180	V80	196	AD0.P0.3
		165	AD0.P0.6	181	V80	197	AD0.P0.4
		166	AD0.P0.7	182	V80	198	AD0.P0.5
		167	AD0.P0.0	183	V80	199	AD0.P0.6
		168	AD0.P0.1	184	V80	200	AD0.P0.7
		169	AD0.P0.2	185	V80	201	AD0.P0.0
		170	AD0.P0.3	186	V80	202	AD0.P0.1
		171	AD0.P0.4	187	V80	203	AD0.P0.2
		172	AD0.P0.5	188	V80	204	AD0.P0.3
		173	AD0.P0.6	189	V80	205	AD0.P0.4
		174	AD0.P0.7	190	V80	206	AD0.P0.5
		175	AD0.P0.0	191	V80	207	AD0.P0.6
		176	AD0.P0.1	192	V80	208	AD0.P0.7
		177	AD0.P0.2	193	V80	209	AD0.P0.0
		178	AD0.P0.3	194	V80	210	AD0.P0.1
		179	AD0.P0.4	195	V80	211	AD0.P0.2
		180	AD0.P0.5	196	V80	212	AD0.P0.3
		181	AD0.P0.6	197	V80	213	AD0.P0.4
		182	AD0.P0.7	198	V80	214	AD0.P0.5
		183	AD0.P0.0	199	V80	215	AD0.P0.6
		184	AD0.P0.1	200	V80	216	AD0.P0.7
		185	AD0.P0.2	201	V80	217	AD0.P0.0
		186	AD0.P0.3	202	V80	218	AD0.P0.1
		187	AD0.P0.4	203	V80	219	AD0.P0.2
		188	AD0.P0.5	204	V80	220	AD0.P0.3
		189	AD0.P0.6	205	V80	221	AD0.P0.4
		190	AD0.P0.7	206	V80	222	AD0.P0.5
		191	AD0.P0.0	207	V80	223	AD0.P0.6
		192	AD0.P0.1	208	V80	224	AD0.P0.7
		193	AD0.P0.2	209	V80	225	AD0.P0.0
		194	AD0.P0.3	210	V80	226	AD0.P0.1
		195	AD0.P0.4	211	V80	227	AD0.P0.2
		196	AD0.P0.5	212	V80	228	AD0.P0.3
		197	AD0.P0.6	213	V80	229	AD0.P0.4
		198	AD0.P0.7	214	V80	230	AD0.P0.5
		199	AD0.P0.0	215	V80	231	AD0.P0.6
		200	AD0.P0.1	216	V80	232	AD0.P0.7
		201	AD0.P0.2	217	V80	233	AD0.P0.0
		202	AD0.P0.3	218	V80	234	AD0.P0.1
		203	AD0.P0.4	219	V80	235	AD0.P0.2
		204	AD0.P0.5	220	V80	236	AD0.P0.3
		205	AD0.P0.6	221	V80	237	AD0.P0.4
		206	AD0.P0.7	222	V80	238	AD0.P0.5
		207	AD0.P0.0	223	V80	239	AD0.P0.6
		208	AD0.P0.1	224	V80	240	AD0.P0.7
		209	AD0.P0.2	225	V80	241	AD0.P0.0
		210	AD0.P0.3	226	V80	242	AD0.P0.1
		211	AD0.P0.4	227	V80	243	AD0.P0.2
		212	AD0.P0.5	228	V80	244	AD0.P0.3
		213	AD0.P0.6	229	V80	245	AD0.P0.4
		214	AD0.P0.7	230	V80	246	AD0.P0.5
		215	AD0.P0.0	231	V80	247	AD0.P0.6
		216	AD0.P0.1	232	V80	248	AD0.P0.7
		217	AD0.P0.2	233	V80	249	AD0.P0.0